Induction, Training, and Parsing Strategies beyond Context-free Grammars

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technische Universität Dresden
Fakultät Informatik

eingereicht am
28. November 2019

von
Dipl.-Inf. Kilian Gebhardt

geboren am 15. 03. 1990
in Friedrichroda

Gutachter:

- Prof. Dr.-Ing. habil. Dr. h.c./Univ. Szeged Heiko Vogler, Technische Universität Dresden
- Prof. Dr.-Ing. Marco Kuhlmann, Lingköpings Universitet

Fachreferent:

- Prof. Dr. Sebastian Rudolph, Technische Universität Dresden

Verteidigt am: 6. Mai 2020 in Dresden
Überarbeite Fassung vom: 19. Juni 2020

Danksagungen

Das letzte Jahr war in vielerlei Hinsicht intensiv. Unter anderem habe ich in dieser Zeit diese Dissertation niedergeschrieben, aber natürlich ging der gedankliche Reifeprozess weit darüber hinaus. An dieser Stelle möchte ich deshalb all den Menschen danken, die mich dabei unterstützt haben.

Ich danke meinem Doktorvater Heiko Vogler für die fachliche Unterstützung und den Freiraum, an meinen eigenen Forschungsideen zu arbeiten. Ich danke Marco Kuhlmann für das Begutachten dieser Dissertation und sein Mitwirken in der Promotionskommission. Mein Dank gilt auch weiteren Forschern an anderen Universitäten: Mark-Jan Nederhof und Frank Drewes für die gemeinsame Arbeit an wissenschaftlichen Artikeln; außerdem Andreas van Cranenburgh für anregende Gespräche und Emailkonversationen sowie Hilfe beim Umgang mit dem Lassy Korpus und `discodop`. Darüber hinaus danke ich meinen Kollegïnnen an der Professur, welche sich stets bereitwillig über Fortschritte und Unwägbarkeiten meiner Forschung informieren ließen, wertvolles Feedback gaben und schlechte Wortwitze ertragen mussten. Danke Markus für die Jahre die wir im Büro geteilt haben und alle Empfehlungen für guten Jazz. Danke Toni für alle Erkenntnisse über den Split/Merge Algorithmus, die du mit mir geteilt hast, und natürlich deine Hilfe bei technischen Problemen jeglicher Couleur. Danke auch an Johannes, Luisa, Tobias, Thomas, Richard und Frederic für wertvolle Hinweise und anregende Gespräche. Danke Kerstin für alle aufmunternden Worte und deine Unterstützung bei organisatorischen Fragen. Besonders möchte ich auch Thomas, Frederic und der mi^3-Gruppe danken, die Entwürfe dieser Arbeit korrekturgelesen und viele Fehler gefunden haben – für die restlichen Fehler stehe ich gerne gerade.

Ich denke es ist mir weitgehend gelungen, die Arbeit und die anderen Seiten des Lebens im Gleichgewicht zu halten. Dabei unterstützt hat mich die mi^3-Gruppe, der ich für alle gemeinsamen Stunden in der Boulderhalle, im Sandstein, beim Frühjahrsklettern oder anderweitiger Geselligkeit danken will. Erwähnen möchte ich an dieser Stelle Andreas, mit dem ich wunderbare Touren im Wetterstein, am Ortler und auf den Lofoten gemacht habe. Für die gemeinsame Zeit in der WG danke ich Helen und Andreas.

Zuletzt gilt besonderer Dank meiner Familie. Danke liebe Großeltern für das interessierte Mitverfolgen und die Unterstützung aus der Ferne. Danke Johanna, Max und Emi, Josefine, und Karin und Martin: ich habe euch lieb.

Contents

1 Introduction **1**

2 Preliminaries **9**
2.1 Mathematical notions 9
2.2 Probability theory 11
2.3 Initial algebra semantics and interpreted regular tree grammars . . . 12
2.3.1 Sorted alphabets, algebras, and homomorphisms 12
2.3.2 Regular tree grammars 16
2.3.3 Initial algebra semantics 18
2.4 Unranked trees and hedges 22
2.5 Hybrid trees 27

3 Training and parsing algorithms for probabilistic IRTG **31**
3.1 Probabilistic interpreted regular tree grammars 32
3.2 Expectation/maximization training for IRTG 38
3.2.1 Inside and outside weights 38
3.2.2 Grammar morphisms 41
3.2.3 The EM-Algorithm 44
3.2.4 Maximum a posteriori estimation with Dirichlet priors 47
3.3 Split/merge algorithm for IRTG 50
3.3.1 Splitting 52
3.3.2 Efficient refinement of a chart 53
3.3.3 Merging 54
3.3.4 Smoothing 57
3.3.5 Split/Merge Cycles 58
3.4 Parsing objectives for probabilistic IRTG 58
3.4.1 NP-hardness of PRTG parsing 59
3.4.2 Viterbi parsing 60
3.4.3 Sampling 61
3.4.4 n-best parsing 61
3.4.5 Reranking n-best derivation trees 62
3.4.6 Projection-based parsing strategies 63

4 Grammar formalisms for discontinuous/non-projective parsing 67
4.1 Linear context-free rewriting systems . . . 67
4.2 Definite clause programs . . . 69
4.3 An alignment algebra for LCFRS and sDCP . . . 74
4.4 LCFRS/sDCP hybrid grammars . . . 84

5 Induction of LCFRS/sDCP hybrid grammars 87
5.1 Decompositions and recursive partitionings . . . 88
5.2 Inducing an LCFRS from a string and a decomposition . . . 91
5.3 Stenciling unranked hedges . . . 94
5.4 Construction of sDCP algebras and sDCPs . . . 96
5.5 Induction of LCFRS/sDCP hybrid grammars . . . 101
5.6 Induction from a corpus . . . 106

6 Charts of hybrid grammars 109
6.1 Charts for LCFRS . . . 110
6.2 Charts for sDCP . . . 113
6.3 Charts for LCFRS/sDCP hybrid grammars . . . 117

7 Discontinuous parsing with split/merge-refined grammars 123
7.1 Requirements of a syntactic parser . . . 124
7.2 Implementation overview . . . 126
7.3 Practical issues . . . 127
7.3.1 Coarse-to-fine parsing for LCFRS with CFG . . . 128
7.3.2 The lexical layer and rare words . . . 129
7.3.3 Part-of-speech tags . . . 130
7.4 Datasets, splits, and statistical significance . . . 131
7.4.1 Statistical significance . . . 131
7.4.2 German corpora . . . 132
7.4.3 Dutch Lassy corpus . . . 133
7.5 Properties of baseline grammars . . . 134
7.6 Split/merge refinement . . . 135
7.6.1 Exploratory analysis for TiGer . . . 137
7.6.2 Results for TiGer (SPMRL split) . . . 142
7.6.3 Results for NeGra . . . 143
7.6.4 Results for Lassy . . . 146
7.6.5 Remarks . . . 147
7.7 Qualitative analysis . . . 147
7.8 External comparison and conclusions . . . 149

8 Comparison with related work **153**
8.1 Transition-based dependency and constituent parsing 157
8.1.1 Pseudo-projective dependency parsing 158
8.1.2 Swap transitions for dependency and constituent parsing . . . 159
8.1.3 Attardi's system 160
8.1.4 Covington parser (SR-GAP) for discontinuous constituent parsing 160
8.2 Graph-based parsing algorithms 161
8.3 Reduction to sequence-to-sequence and sequence labeling tasks . . . 162
8.4 Pseudo-projective grammar-based approaches. 163
8.5 Related work on LCFRS parsing 164
8.5.1 Discontinuous data-oriented parsing 166
8.5.2 Inclusion of function labels 167
8.6 Related work on grammars with latent annotations. 167

9 Conclusions **171**

Bibliography **175**

A Appendix **205**
A.1 Appendices to Chapter 3 205
A.2 Appendices to Chapter 4 205
A.3 Proof of Lemma 5.3.3 212

Index **215**

1 Introduction

A distinguishing feature of humankind is the capacity to use symbolic language for communication and representation of information. Computers, which are meant to process information at scale, have been envisioned to process *natural language*, i.e., the language spoken and written by humans, early on (Turing 1950). The potential of this vision is evident by applications that entered our daily life, such as speech recognition (Benesty, Sondhi, and Y. Huang 2007), automatic translation (Lopez 2008), and spell checking (B. Martins and Silva 2004). *Natural language processing* (NLP) is a scientific field that investigates the methodology underlying such systems. Although it might be possible to build monolithic algorithms that implement any of the applications mentioned above, in the past research has favored modular approaches (Bird and Loper 2004; Manning et al. 2014), which address different layers of language, such as phonetics, morphology, syntax, semantics, and pragmatics (Bender 2013, Table 1.1), individually. Modularity not only divides the problem into smaller, more manageable pieces but also facilitates the scientific analysis, which can be performed individually for each layer. In this thesis, we consider the problem of analyzing written sentences syntactically. Syntax is the structure of a language and, intuitively speaking, constitutes the difference between a sentence and a bag of words (Bender 2013). For many tasks, such as machine translation, information extraction, and sentiment analysis, structural information conveying "who did what to whom" is beneficial (Culotta and Sorensen 2004; Ding and Palmer 2005; Duric and Song 2011) and anticipates the actual meaning.

Early investigations of syntax date back to the antiquity (e.g., Pāṇini, 400 BCE, or Dionysius Thrax, 170–90 BCE). Modern syntax was highly influenced by Chomsky (1956, 1959), who proposed rule-based devices to generate the set of all sentences of a language. The derivational process of such a *grammar* for a particular sentence can be represented as a so-called *parse tree*, which in turn is perceived as the syntactic analysis of the sentence. We call the process of assigning a sentence its parse tree *parsing*. One grammar formalism introduced by Chomsky are the well-known *context-free grammars* (CFG), which were soon found to be too weak to model certain phenomena occurring in natural languages. For instance, the so-called Wh-movement in English questions (see Figure 1.1 and Sag 2010 for an overview), which dislocates the question word from its phrasal context, requires a transformation of the CFG parse tree. Dutch and Swiss-German allow in principle for unlimited cross-serial dependencies (Shieber

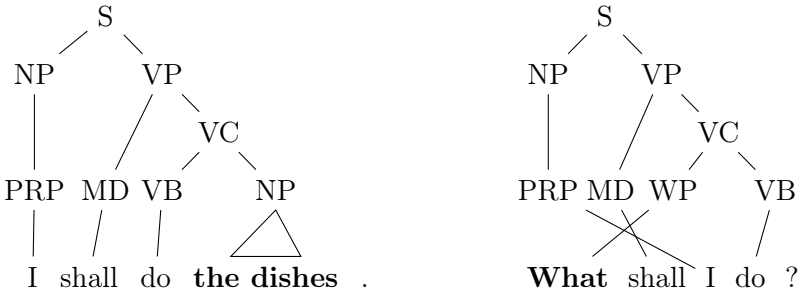


Figure 1.1: Wh-movement in an English question analyzed as a discontinuous parse tree.

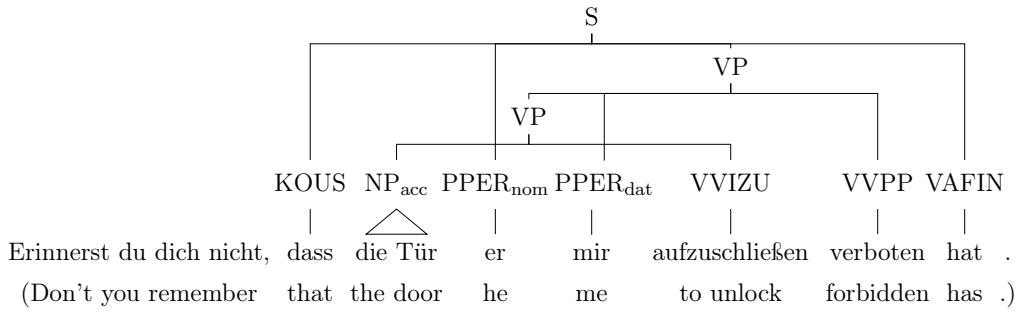


Figure 1.2: Unbounded scrambling in a German sentence *(Don't you remember that he has forbidden me to unlock the door.)* with a discontinuous parse tree. Both nodes labeled VP (meaning *verb phrase*) do not cover a continuous sequence of sentence positions.

1985), which can abstractly be formalized as a language $\{wf(w) \mid w \in \Sigma^*\}$ where Σ is an alphabet and f is a string homomorphism. Such a *copy language* is, in general, not context-free. Lastly, due to a rich morphology, German permits a high degree of flexibility in how the arguments of the verb are ordered (Becker, Niv, and Rambow 1992) in subordinate clauses, a phenomenon called unbounded scrambling. Figure 1.2 gives an example. Note that the argument "die Tür" of "aufzuschließen" is interleaved by "er" and "mir", which are not arguments of "aufzuschließen". To model these phenomena, syntactic analyses were proposed where the nodes in parse trees may cover sequences of sentence positions that are not continuous. Such parse trees are called *discontinuous* (McCawley 1982; Stucky 1987). Internal nodes that cover a discontinuous interval are called *discontinuous constituents* (e.g., both nodes labeled VP in Figure 1.2). In the German TiGer corpus (Brants et al. 2004), a collection of ca. 50,000 syntactically annotated sentences, 27.8% of the sentences have discontinuous

parse trees and 4.8% of all constituents are discontinuous.[1]

Different options to account for such phenomena have been studied and formalized to a varying degree. The next more expressive class of languages in the Chomsky hierarchy are the context-sensitive languages. Their expressiveness comes at the cost of increased algorithmic complexity. As a trade-off between expressiveness and complexity, the notion of *mild context-sensitivity* (Joshi 1985) has been coined: a grammar formalism is mildly context-sensitive if it extends the context-free languages, it is capable of modeling a restricted amount of cross-serial dependencies, each rule generates only a limited number of symbols, and there is an algorithm that recognizes if a grammar generates a given sentence in time polynomial in the length of the sentence. Some members of this class of grammar formalisms are the *tree adjoining grammars* (TAG, Joshi, L. S. Levy, and Takahashi 1975), the *combinatory categorial grammars* (CCG, Ades and Steedman 1982; Szabolcsi 1989), and the *linear context-free rewriting systems* (LCFRS, Vijay-Shanker, Weir, and Joshi 1987).

Although the mildly context-sensitive grammar formalisms have a polynomial parsing complexity, processing long sentences with, for instance, an $O(n^{12})$ algorithm is often prohibitive in practice. Hence, many early parsers, i.e., algorithms for syntactic analysis, opt for less expressive but feasible grammar formalisms (M. J. Collins 1999; Charniak 2000; Klein and Manning 2003). Some approaches to retain at least a part of the discontinuous constituents apply a post-processing step after parsing with a CFG (Johnson 2002; Dienes and Dubey 2003; R. Levy and Manning 2004; Schmid 2006; Boyd 2007; Cai, Chiang, and Goldberg 2011; Versley 2016). A recent advance in this direction is hybrid grammars (Nederhof and Vogler 2014; Gebhardt, Nederhof, and Vogler 2017), which couple a not necessarily mildly context-sensitive string grammar with a tree generating device: a sentence is generated by the string grammar while a tree is generated synchronously by the tree grammar. The structure of the tree can be very different from that of the generative process. This allows discontinuous analyses even if a regular string grammar, which has linear parsing complexity, is used. Another way to tackle the high parsing complexity is by discarding exact parsing in favor of faster approximate strategies. LCFRS admits a context-free approximation (Barthélemy et al. 2001) that can be utilized to prune the search space for the LCFRS parsing step (Burden and Ljunglöf 2005; van Cranenburgh 2012). More recently, parsing approaches without a grammar have been proposed: a transition system reads the sentence from left to right and makes use of hand-picked actions to incrementally combine nodes to a parse tree. Given the capability to reorder the nodes in its internal data structures (Versley 2014a; Maier 2015; Coavoux and Crabbé 2017) or to combine non-adjacent nodes (Coavoux and Cohen 2019), the system can produce discontinuous analyses. A discriminative classifier is trained to greedily select the appropriate actions, which results in a low run-time of the parser.

An orthogonal problem to the complexity of discontinuous parsing is the ambiguity

[1] Measured with `treetools` after removing spurious discontinuity due to punctuation.

of language, i.e., a sentence may admit multiple plausible analyses. For instance, in the sentence "She saw the astronomer with the telescope." the prepositional phrase "with the telescope" might be attached to "saw" or to "the astronomer". In the former case, the act of "seeing" is mediated by "the telescope". In the latter case, "the astronomer" has "the telescope". Although most ambiguity can be resolved if the context of a sentence and world knowledge is taken into account, such an approach is beyond the modular setting that we consider. Instead, we resolve the non-determinism by extending the formal grammars with a weight structure that allows us to assign to each analysis of a given sentence a weight. An example of such a weight structure is the probabilistic semiring, which enables the grammar to assign each analysis a probability. Probabilistic grammars and algorithms to utilize them have been studied intensively in the last 50 years (e.g., Suppes 1972; Baker 1979; Lari and Young 1990; Nederhof 2003; Nederhof and Satta 2004; Corazza and Satta 2006).

The probabilistic approach usually requires a dataset from which the grammar's weights are estimated. If also the grammar itself is aggregated from data automatically, then we speak of *data-driven parsing*. A challenging aspect of this process is the generalization of structural patterns found in the data. First, one needs to identify the patterns. This phase is also called *grammar induction* and needs to be customized to the grammar formalism. Each tree in a treebank, i.e., a selection of syntactically annotated sentences, is hierarchically decomposed into smaller parts. A rule is created for each of these parts with the responsibility to generate it. Depending on the grammar formalism, these rules work by rewriting nonterminals to some terminal structure and further nonterminals. For CFG, so-called treebank grammars (Charniak 1996) can be read off continuous parse trees. LCFRS can be obtained from treebanks with discontinuous parse trees via a generalization of this method (Maier and Søgaard 2008). Nederhof and Vogler (2014) and Gebhardt, Nederhof, and Vogler (2017) present multiple induction algorithms for hybrid grammars where the parsing complexity can be controlled.

Assuming that a method for grammar induction has been found, we turn to the problem that rules shall be applicable only under certain side conditions, which are typically encoded into the nonterminals. Isolating these side conditions in an automated fashion is preferable, albeit challenging. Early work, e.g., by M. J. Collins (1999), puts much effort into the design of conditional contexts of the various generative processes of their parsers. Klein and Manning (2003) present a technique called Markovization, where nonterminal symbols are enhanced by a limited vertical and horizontal context. This context information is comprised of symbols occurring above and in sibling positions to the symbol in the tree from which the rule is obtained. The parameterization of the size of the vertical and horizontal context is meant to be equal for all rules of the grammar. However, it is rather unlikely that the same size is optimal for each rule. Instead of hard-coding the context information into the nonterminals of the grammar, Matsuzaki, Miyao, and Tsujii (2005) propose an extension of *probabilistic context-free grammars* (PCFG) where the assessment of

applicability is shifted into the weight structure. In this approach, the unweighted grammar encodes only weak restrictions on the applicability of rules. The weight structure is chosen such that the plausible analyses are preferred. This is achieved by associating a hidden state to each occurrence of a nonterminal. This hidden state, also called *latent annotation*, ranges over a set of k possible states. The distribution over the set of applicable rules is customized for each pair of nonterminal and latent annotation. Petrov et al. (2006) refine this approach by allowing different numbers of latent annotations for different nonterminals. Moreover, they present an algorithm that adaptively increases the number of latent annotations starting from a grammar without latent annotations.

While the majority of this work is focused on PCFG, one may also extend other grammar and transducer formalisms with weight structures. In this context, the question arises if the theory developed for PCFG holds for other formalisms as well. Distilling a common core of many grammar and transducer formalisms will help us to answer this question. Initial algebra semantics (Goguen et al. 1977) advocates the specification of formal languages through a language of trees over operator symbols and an algebra in which the operator trees are evaluated. Based on this idea, Koller and Kuhlmann (2011) propose to frame various grammar and transduction devices as *interpreted regular tree grammars* (IRTG). An IRTG consists of a *regular tree grammar* (RTG, Brainerd 1968; Brainerd 1969; Thatcher and Wright 1968) and a finite number of algebras that interpret the operator trees generated by the RTG. Many grammar and transducer formalisms such as CFG, LCFRS, TAG, synchronous context-free grammars (Chiang 2007), hyperedge replacement grammars (Habel 1992), and extended top-down tree transducers (Arnold and Dauchet 1975) can be represented as IRTG. Moreover, if we instantiate the probabilistic extension of IRTG for, e.g., CFG and LCFRS, then we obtain probabilistic CFG and probabilistic LCFRS, respectively.

Drewes, Gebhardt, and Vogler (2016) characterized hybrid grammars, which pair LCFRS and *simple definite clause programs* (sDCP, Deransart and Małuszyński 1985 and 1989), in the IRTG framework by representing them as particular graph grammars. In this thesis, we maintain an IRTG-view on hybrid grammars. However, we give an alternative representation that avoids graphs and views LCFRS and sDCP as IRTGs with algebras that operate over tuples of strings and trees, respectively. Moreover, in order to model the crucial synchronization between symbols in the string and the tree component of the hybrid grammar, we introduce a novel alignment algebra.

IRTG facilitate the development of generic algorithms for, e.g., binarization (Büchse, Koller, and Vogler 2013) and coarse-to-fine parsing (Teichmann, Koller, and Groschwitz 2017). In this thesis, generic versions of Petrov et al.'s (2006) algorithm and various parsing objectives are introduced. We suppose that the IRTG framework is particularly suitable for a specification of this algorithm because the nonterminals of the RTG are invisible if operator trees are interpreted in the algebras.

Hence, states in the RTG may be regarded latent *a priori*. In other words, in IRTG, latent annotations come for free.

We are particularly interested in the application of the generalization of Petrov et al.'s (2006) algorithm to the problem of discontinuous parsing with LCFRS and hybrid grammars. We hypothesize that the algorithm yields grammars with a good trade-off between accuracy and coverage when applied to syntactic parsing. Moreover, we analyze how the parameterization of the algorithm influences the properties of the resulting grammars. We compare these grammars to related approaches, in particular, vanilla LCFRS and hybrid grammars, the discontinuous data-oriented parsing model, transition-based models for discontinuous parsing, and pseudo-projective approaches.

Thus, the plan for this thesis is as follows: First, we outline the notation that we use in this thesis in Chapter 2. In particular, we define IRTG over many-sorted algebras (Birkhoff and Lipson 1970), hedges (i.e., a data structure to represent sequences of trees) over unranked alphabets, and hybrid trees (to represent discontinuous parse trees). In Chapter 3, we develop the probabilistic extension of the IRTG framework, the generalization of Petrov et al.'s (2006) algorithm, and different approaches to use IRTGs for parsing. The grammar formalisms which we make use of in this thesis are defined as particular IRTGs in Chapter 4. Specifically, we present a definition for LCFRS, for sDCP, and for hybrid grammars that couple LCFRS and sDCP. Algorithms to construct hybrid grammars from corpora such as TiGer are specified in Chapter 5. The generalization of Petrov et al.'s algorithm requires a compact representation of the set of all derivations that the grammar has for a particular object. How such compact representations can be obtained for LFCRS, sDCP, and hybrid grammars, is dealt with in Chapter 6. This concludes the theoretic part of this thesis.

In Chapter 7, we analyze the effect of Petrov et al.'s algorithm on LCFRS and hybrid grammars constructed from different treebanks. We also consider various hyperparameters of the induction and the influence of the parsing objective. The parsing performance of the different grammars is compared to models from the literature. Chapter 8 gives an overview of these related approaches and how they compare our grammars.

Arguably, the scope of this thesis is in between theoretical computer science and practical NLP. This hybrid status poses the challenge to do justice to both fields, which we certainly will not achieve fully. On the one hand, we give a formal specification of the machinery involved and provide algorithms that are applicable beyond the problem of discontinuous constituent parsing that we try to solve in the applied part. However, we present only a limited number of theoretical results. In particular, we do not compare the expressiveness of hybrid grammars to that of other grammar formalisms. Also, we do not formally show the correctness of some statements, which are mostly motivated by practical application but would be very involved to prove. On the other hand, we devote a big part of this work to the implementation and execution of an empirical analysis of hybrid grammars to constituent parsing. Still,

there are obvious variations to the experimental setup that we did not perform. Moreover, we leave a deep analysis of the grammars that could be learned for a range of topologically diverse languages open for further investigation.

2 Preliminaries

2.1 Mathematical notions

We denote the operations *union*, *intersection*, and *difference* on sets by $\cup$, $\cap$, and $\setminus$, respectively. We denote the set of non-negatives integers by $\mathbb{N}$ and the set of intergers by $\mathbb{Z}$. Let $n, m \in \mathbb{Z}$ with $n \leq m$. We denote by $[n, m]$ the set $\{n, n+1, \ldots, m\}$. As short-hand we use $[n]_0 = [0, n]$ and $[n] = [1, n]$. Observe that for each $n \leq 0$ we have $[n] = \emptyset$ and that for each $n < 0$ we have $[n]_0 = \emptyset$. We denote the set of real numbers by $\mathbb{R}$ and define $[0, 1]_{\mathbb{R}} = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$. A finite, non-empty set is called *alphabet*.

Let X, Y, and Z be sets. The *powerset of* X, i.e., $\{X' \mid X' \subseteq X\}$, is denoted by $\mathfrak{P}(X)$. A *partial function* f from X to Y is a subset of $X \times Y$ where for each $x \in X$ there is at most one element $y \in Y$ such that $(x, y) \in f$. If such an element y exists, it is denoted by $f(x)$. We may write $f \colon X \dashrightarrow Y$ to indicate that f is a partial function from X to Y. We call f *injective* if, for all $x, x' \in X$ and $y \in Y$, $\{(x, y), (x', y)\} \subseteq f$ implies $x = x'$. We call f *surjective* if, for each $y \in Y$, there exists $x \in X$ such that $(x, y) \in f$. We call f *bijective* if f is injective and surjective.

Let $f \colon X \dashrightarrow Y$. If for each $x \in X$ there is $y \in Y$ such that $(x, y) \in f$, then f is a (proper) function, indicated by $f \colon X \to Y$. We denote the set of all functions from X to Y by $X \to Y$. Moreover, we may refer to the domain X by $\mathrm{dom}(f)$ and to the codomain Y by $\mathrm{codom}(f)$. Sometimes we specify f as a set using the notation $\{x_1 \mapsto y_1, x_2 \mapsto y_2, \ldots\}$, where, for each i, $y_i = f(x_i)$. Notably, if $\mathrm{dom}(X) = \emptyset$, then $f = \emptyset$. By $\mathrm{id}_X \colon X \to X$ we denote the *identity function*, i.e., for each $x \in X$, $\mathrm{id}_X(x) = x$.

Let $f \colon X \to Y$ and $g \colon Y \to Z$. The *composition of* f *and* g is the function $g \circ f \colon X \to Z$ such that $(g \circ f)(x) = g(f(x))$ for each $x \in X$. We assume that the function arrow $\to$ in type specifications is right-associative, e.g., $f \colon A \to B \to C$ is read $f \colon A \to (B \to C)$. We may lift f to a function $f' \colon \mathfrak{P}(X) \to \mathfrak{P}(Y)$ such that $f'(X) = \{f(x) \mid x \in X\}$ but denote f' by f too. We define the *inverse of* f to be the function $f^{-1} \colon Y \to \mathfrak{P}(X)$ such that $f^{-1}(y) = \{x \in X \mid f(x) = y\}$. If $h \colon X \to X$, then we define, for each $i \in \mathbb{N}$, the function $h^i \colon X \to X$ such that $h^0 = \mathrm{id}_X$ and $h^{i+1} = h^i \circ h$.

Let I be a countable set and B be a class[1]. An *I-indexed family* is a function $A\colon I \to B$. For convenience, we may write A_i instead of $A(i)$ and may specify A by writing $A = (A_i \mid i \in I)$. We may say *A is an I-indexed family of sets (functions)* to express that each A_i is a set (a function).

Let X be a set, $n \in \mathbb{N}$, and for each $i \in [n]$, let $X_i \subseteq X$. We call $(X_i \mid i \in [n])$ a *partition of X*, if (i) for each $i \in [n]$, $X_i \neq \emptyset$, (ii) for each $i, j \in [n]$ with $i \neq j$, $X_i \cap X_j = \emptyset$, and (iii) $X = \bigcup_{i\in[n]} X_i$.

Strings. Let X be a set. A *(finite) string over X* is a sequence w of the form $x_1 \dots x_n$ where $n \in \mathbb{N}$ and, for each $i \in [n]$, we have that $x_i \in X$. We denote the *length n of w* by $|w|$ and use the notation $w[i]$ to refer to the symbol x_i. Further, $w[i..j]$ denotes the substring $x_i x_{i+1} \cdots x_j$. In particular, if $j < i$, then $w[i..j]$ is the empty string. We denote the set of all positions of w by $[|w|]$. The set of all finite strings over X is denoted by X^*. The empty string is denoted by ε. If $w, u \in X^*$, then $w \cdot u$ (short: wu) denotes the *concatenation of w and u*. Concatenation is lifted to sets of strings where $W \cdot U = \{w \cdot u \mid w \in W, u \in U\}$ as usual. In particular, $W \cdot \emptyset = \emptyset = \emptyset \cdot W$ for every set $W \subseteq \Sigma^*$. We may abuse notation and write $x \in w$, if there exists $i \in [|w|]$, such that $w[i] = x$.

Relations and orderings. Let A be a set. A *relation on A* is a subset of $A \times A$. We denote the *identity relation* on A by $\mathrm{id}_A = \{(a, a) \mid a \in A\}$[2]. Let $\bowtie \in A \times A$ and $a_1, a_2 \in A$. If $(a_1, a_2) \in \bowtie$, then we may write $a_1 \bowtie a_2$. $\bowtie$ is called *reflexive* if $\mathrm{id}_A \subseteq \bowtie$. $\bowtie$ is called *irreflexive*, if $\mathrm{id}_A \cap \bowtie = \emptyset$. $\bowtie$ is called *transitive*, if for all $a_1, a_2, a_3 \in A$: $a_1 \bowtie a_2$ and $a_2 \bowtie a_3$ implies $a_1 \bowtie a_3$. $\bowtie$ is called *symmetric* if for all $a_1, a_2 \in A$: $a_1 \bowtie a_2$ implies $a_2 \bowtie a_1$. $\bowtie$ is called *anti-symmetric* if for all $a_1, a_2 \in A$: $a_1 \bowtie a_2$ and $a_2 \bowtie a_1$ implies $a_1 = a_2$. The *reflexive closure of* $\bowtie$, denoted by $\underline{\bowtie}$, is defined as $\underline{\bowtie} = \bowtie \cup \mathrm{id}_A$. The *reflexive and transitive closure of* $\bowtie$, denoted by $\underline{\bowtie}^{\circledast}$, is defined such that $\underline{\bowtie}^{\circledast} = \bigcup_{n\in\mathbb{N}} \bowtie^n$, where $\bowtie^0 = \mathrm{id}_A$ and $\bowtie^{i+1} = \{(a, c) \mid \exists b \in A\colon a \bowtie^i b \wedge b \bowtie c\}$ for each $i \in \mathbb{N}$. If $\bowtie$ is reflexive, transitive, and anti-symmetric, then $\bowtie$ is called *partial order*. If $\bowtie$ is irreflexive, transitive, and anti-symmetric, then $\bowtie$ is called *strict order*. If $\bowtie$ is a partial order, and for each $a_1, a_2 \in A$ we have $a_1 \bowtie a_2$ or $a_2 \bowtie a_1$, then $\bowtie$ is a *total order*. Let $\bowtie$ be a partial order on A. The *lexicographic order induced by* $\bowtie$, denoted by $\bowtie_{\mathrm{lex}}$, is the partial order on A^* such that $\varepsilon \bowtie_{\mathrm{lex}} w$ for each $w \in A^*$ and, for $a_1, a_2 \in A$ and $w_1, w_2 \in A^*$, it holds that $a_1 w_1 \bowtie_{\mathrm{lex}} a_2 w_2$ if ($a_1 \bowtie a_2$ and $a_1 \neq a_2$) or ($a_1 = a_2$ and $w_1 \bowtie_{\mathrm{lex}} w_2$).

Substitution. Let X, Y, and Z be sets such that X and Y are disjoint and let $\varphi\colon Y \to Z$. Let $w \in (X \cup Y)^*$. The string in $(X \cup Z)^*$ obtained by substituting in w

[1] In particular B may be the class of all sets.

[2] The identity function and the identity relation are the same object. Hence it is safe to use the same identifier.

each occurrence of $y \in Y$ with $\varphi(y)$ is denoted by $w[y/\varphi(y) \mid y \in Y]$.

Variables. Throughout this thesis, we will make use of variables. The set X contains variables that are indexed twice, i.e., $X \subseteq \{x_j^i \mid i \in \mathbb{N}, j \in \mathbb{N}\}$. The set Y will be used for variables that are indexed once, i.e., $Y \subseteq \{y_i \mid i \in \mathbb{N}\}$. It is assumed that variables with different indices are distinct, e.g., $x_j^i = x_k^l$ implies $i = l$ and $j = k$. We may denote the set $\{y_1, \ldots, y_n\}$ by Y_n for every $n \in \mathbb{N}$.

2.2 Probability theory

In this section we introduce notions to deal with probability theory. As most of the time we only consider distributions over discrete sets, here we only present a simplified version of probability theory avoiding measure theory and σ-algebras.

Let X be a set. A *probability distribution over* X is a function $p\colon X \to [0,1]_{\mathbb{R}}$ such that $1 = \sum_{x \in X} p(x)$. We denote the set of all probability distributions over X by $\mathcal{M}(X)$. Let X and Y be sets. A *conditional probability distribution over* Y *given* X is a function $p\colon X \to (Y \to [0,1]_{\mathbb{R}})$ such that $p(x)$ is a probability distribution over X for each $x \in X$. For each $x \in X$ and $y \in Y$ we also write $p(y \mid x)$ instead of $(p(x))(y)$.

Let $n \in \mathbb{N}$, $f\colon X \to \mathbb{R}^n$, and $p \in \mathcal{M}(X)$. The *expected value of* f *given* p is

$$\mathbb{E}_p[f] = \sum_{x \in X} p(x) \cdot f(x) \ ,$$

where "$\cdot$" denotes scalar multiplication and vector addition is component-wise. We make use of lambda abstractions when specifying expected values, e.g., if $g\colon B \times C \to \mathbb{R}$ and $c \in C$, then $\lambda b.g(b,c)$ denotes the function $g'\colon B \to \mathbb{R}$ such that $g'(b) = g(b,c)$.

Let $p, q \in \mathcal{M}(X)$ be such that for every $x \in X$, $q(x) = 0$ implies $p(x) = 0$. The *Kullback–Leibler (KL) divergence between* q *and* p, denoted by $\mathrm{KL}(p \mid\mid q)$, is defined by

$$\mathrm{KL}(p \mid\mid q) = \sum_{x \in X} p(x) \log\left(\frac{p(x)}{q(x)}\right) \ ,$$

where we assume that $p(x)\log\left(\frac{p(x)}{q(x)}\right) = 0$ if $p(x) = 0$ for some $x \in X$.[3] Two essential properties of the KL divergence are

$$\mathrm{KL}(p \mid\mid q) \geq 0$$

and

$$\mathrm{KL}(p \mid\mid q) = 0 \quad \text{if and only if} \quad p = q \ .$$

[3] This assumption is reasonable because $\lim_{z \to 0^+} z \cdot \log(z) = 0$.

A *corpus over* X is a function $c\colon X \to \mathbb{R}_{\geq 0}$. Let $o = x_1, x_2, x_3, \ldots$ be a sequence of observations and $p \in \mathcal{M}(X)$. The probability $\mathrm{P}(o \mid p)$ with which o was drawn from p is called *likelihood of o under p*. If we assume that the single events x_i in o are *independent of another and identically distributed* (i.i.d.), then we can represent o as a corpus c by setting $c(x)$ to the number of occurrences x has in o and obtain

$$\mathrm{P}(c \mid p) = \mathrm{P}(x_1 \mid p) \cdot \mathrm{P}(x_2 \mid p) \cdot \ldots = \prod_{x \in X} \mathrm{P}(x \mid p)^{c(x)} = \prod_{x \in X} p(x)^{c(x)} \ .$$

In the following, we always use this definition of likelihood and apply it to any corpus (and not just those obtained by a conversion of a sequence of i.i.d. observations). Instead of the likelihood, we may also use the *log-likelihood*, i.e.,

$$\log \mathrm{P}(c \mid p) = \sum_{x \in X} c(x) \cdot \log(p(x)) \ .$$

If c is also a probability distribution (and in fact, we can often normalize it to be one), then minimizing the KL divergence between c and p is the same as maximizing the likelihood of c under p.

2.3 Initial algebra semantics and interpreted regular tree grammars

2.3.1 Sorted alphabets, algebras, and homomorphisms

We recall notions from Birkhoff and Lipson (1970) in order to define sorted (or typed) algebras and homomorphisms between them. Intuitively, an algebra is the combination of a domain or carrier set with a finite set of operations that act on this domain. A sorted algebra is a particular algebra whose domain is partitioned into different subdomains, e.g., integers and Boolean values. Its operations are typed, i.e., they expect a fixed sequence of arguments, each of which needs to be in a particular subdomain, and produce a result in a particular subdomain. For instance, the operation "+" might require two integers as arguments and yield an integer, whereas the operation "<" requires two integers as arguments and yields a Boolean value as a result. These notions are formalized in the following, where we adhere to the convention of calling a type *sort*.

Definition 2.3.1. Let S be a countable set. An *S-sorted set* is pair (Z, sort) where Z is a set and $\mathrm{sort}\colon Z \to S$. (Z, sort) is an *S-sorted alphabet* if Z is an alphabet. We may abbreviate (Z, sort) by Z. For each $s \in S$, we denote the set $\{z \in Z \mid \mathrm{sort}(z) = s\}$ by Z_s. □

In examples we may specify an S-sorted alphabet (Z, sort) by $Z = \{z_1^{(s_1)}, \ldots, z_n^{(s_n)}\}$ if $Z = \{z_1, \ldots, z_n\}$ and $\mathrm{sort}(z_i) = s_i$ for each $i \in [n]$.

Sorted sets are particularly interesting for defining trees (or terms) over operator symbols which shall be evaluated in a type-safe manner. These trees need to adhere the sort-constraints of the operator symbols. In the following, we call said trees over operator symbols *operator trees*.

Definition 2.3.2. Let S be a countable set, Σ be an $S^* \times S$-sorted alphabet, and let H be an S-sorted set. The *set of S-sorted trees over Σ* indexed by H, denoted by $\mathrm{T}_\Sigma(H)$, is the smallest S-sorted set T such that

- $H_s \subseteq T_s$, and
- for each $\sigma \in \Sigma_{(s_1 \dots s_k, s)}$ with $s, s_1, \dots, s_k \in S$, $t_1 \in T_{s_1}, \dots, t_k \in T_{s_k}$, we have $\sigma(t_1, \dots, t_k) \in T_s$.

If $H_s = \emptyset$ for each $s \in S$, then we write T_Σ instead of $\mathrm{T}_\Sigma(H)$. □

Definition 2.3.3. Let S be a countable set, Σ be an $S^* \times S$-sorted alphabet, and H be an S-sorted set. Let $s \in S$ and $t \in \mathrm{T}_\Sigma(H)_s$. The set of *positions of t* (also called *Gorn addresses*; cf. Gorn 1967), denoted by $\mathrm{pos}(t)$, is defined recursively such that

$$\mathrm{pos}(t) = \begin{cases} \{\varepsilon\} & \text{if } t \in H_s \\ \{\varepsilon\} \cup \{iw \mid i \in [k], w \in \mathrm{pos}(t_i)\} & \text{if } t = \sigma(t_1, \dots, t_k) \\ & \text{for } k \in \mathbb{N},\ s_1, \dots, s_k \in S, \\ & \sigma \in \Sigma_{(s_1 \cdots s_k, s)}, \text{ and} \\ & t_1 \in \mathrm{T}_\Sigma(H)_{s_1}, \dots, t_k \in \mathrm{T}_\Sigma(H)_{s_k} \ . \end{cases}$$

Let now $w \in \mathrm{pos}(t)$. The *label of t at position w*, denoted by $t(w)$, is defined such that

$$t(w) = \begin{cases} t & \text{if } t \in H_s \text{ and } w = \varepsilon \\ \sigma & \text{if } t = \sigma(t_1, \dots, t_k) \text{ for } k \in \mathbb{N},\ s_1, \dots, s_k \in S,\ \sigma \in \Sigma_{(s_1 \cdots s_k, s)}, \\ & t_1 \in \mathrm{T}_\Sigma(H)_{s_1}, \dots, t_k \in \mathrm{T}_\Sigma(H)_{s_k}, \\ & \text{and } w = \varepsilon \\ t_i(w') & \text{if } t = \sigma(t_1, \dots, t_k) \text{ for } k \in \mathbb{N},\ s_1, \dots, s_k \in S,\ \sigma \in \Sigma_{(s_1 \cdots s_k, s)}, \\ & t_1 \in \mathrm{T}_\Sigma(H)_{s_1}, \dots, t_k \in \mathrm{T}_\Sigma(H)_{s_k}, \\ & \text{and } w = iw' \text{ for } i \in [k] \text{ and } w' \in \mathrm{pos}(t_i) \ . \end{cases}$$

Let $U \subseteq \Sigma \cup H$. We define $\mathrm{pos}_U(t) = \{p \in \mathrm{pos}(t) \mid t(p) \in U\}$. □

Although we mainly use sorted algebras to define formal languages later on, let us continue with an introductory example where we model integer and Boolean expressions.

Example 2.3.4. Let $S = \{\mathrm{b}, \mathrm{i}\}$. Let H be an S-sorted set where $H_\mathrm{b} = \{\top, \bot\}$ and $H_\mathrm{i} = \mathbb{N}$. Let Σ be an $S^* \times S$-sorted set containing the following elements where sorts are annotated in superscript: $\{\oplus^{(\mathrm{ii},\mathrm{i})}, \otimes^{(\mathrm{ii},\mathrm{i})}, \circledcirc^{(\mathrm{ii},\mathrm{b})}, \mathrm{ite}^{(\mathrm{bii},\mathrm{i})}\}$. A tree in $\mathrm{T}_\Sigma(H)_\mathrm{b}$ is $\mathrm{ite}(\circledcirc(1,2), \oplus(3,4), \otimes(2,2))$. □

Definition 2.3.5 (Σ-algebra)**.** Let S be a countable set and let Σ be an $S^* \times S$-sorted set. A *Σ-algebra* is a tuple $(\mathcal{A}, \cdot^\mathcal{A})$ where

- $\mathcal{A}$ is an S-indexed family of sets (called *domain*), and
- $\cdot^\mathcal{A} \colon \Sigma \to \left(\bigcup_{k \in \mathbb{N}} \left(\left(\bigcup_{s \in S} \mathcal{A}_s \right)^k \to \left(\bigcup_{s \in S} \mathcal{A}_s \right) \right) \right)$ maps each symbol to an *operation*

such that for each $k \in \mathbb{N}$, $s_0, \ldots, s_k \in S$, and $\sigma \in \Sigma_{(s_1 \cdots s_k, s_0)}$ it holds that

$$\cdot^\mathcal{A}(\sigma) \colon \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_k} \to \mathcal{A}_{s_0} \ .$$

We denote $\cdot^\mathcal{A}(\sigma)$ by $\sigma^\mathcal{A}$ in the following, and abbreviate $(\mathcal{A}, \cdot^\mathcal{A})$ by $\mathcal{A}$. □

An important example for a Σ-algebra is the Σ-term algebra, which has trees as domain and each symbol $\sigma \in \Sigma_{(s_1 \cdots s_k, s_0)}$ is assigned the operation that extends the k input trees by σ.

Definition 2.3.6. Let S be a countable set, Σ an $S^* \times S$-sorted set, and H an S-sorted set. The *Σ-term algebra indexed by H*, denoted by $\mathrm{T}_\Sigma(H)$, is the Σ-algebra $(\mathcal{A}, \cdot^\mathcal{A})$ where

- $\mathcal{A}_s = (\mathrm{T}_\Sigma(H))_s$ for each $s \in S$ and
- for each $\sigma \in \Sigma_{(s_1 \cdots s_k, s)}$ with $k \in \mathbb{N}$, $s, s_1, \ldots, s_k \in S$, and $t_1 \in (\mathrm{T}_\Sigma(H))_{s_1}$, …, $t_k \in (\mathrm{T}_\Sigma(H))_{s_k}$, we have $\sigma^\mathcal{A}(t_1, \ldots, t_k) = \sigma(t_1, \ldots, t_k)$.

Again, if $H_s = \emptyset$ for each $s \in S$, then we also write T_Σ instead of $\mathrm{T}_\Sigma(H)$. □

Definition 2.3.7 (Σ-homomorphism)**.** Let S be a countable set, Σ be an $S^* \times S$-sorted alphabet, and $(\mathcal{A}, \cdot^\mathcal{A})$ and $(\mathcal{B}, \cdot^\mathcal{B})$ be Σ-algebras. The S-indexed family $\varphi = (\varphi_s \colon \mathcal{A}_s \to \mathcal{B}_s \mid s \in S)$ is a *Σ-homomorphism from $(\mathcal{A}, \cdot^\mathcal{A})$ to $(\mathcal{B}, \cdot^\mathcal{B})$* if, for each $k \in \mathbb{N}$, $s_1, \ldots, s_k, s \in S$, $\sigma \in \Sigma_{(s_1 \ldots s_k, s)}$, and $a_1 \in \mathcal{A}_{s_1}, \ldots, a_k \in \mathcal{A}_{s_k}$, it holds that

$$\varphi_s(\sigma^\mathcal{A}(a_1, \ldots, a_k)) = \sigma^\mathcal{B}(\varphi_{s_1}(a_1), \ldots, \varphi_{s_k}(a_k)) \ .$$ □

Proposition 2.3.8 (Birkhoff and Lipson 1970, Prop. 15)**.** Let S be a countable set, Σ be an $S^* \times S$-sorted alphabet, H be an S-sorted set, and $(\mathcal{A}, \cdot^\mathcal{A})$ be a Σ-algebra. Let $\varphi = (\varphi_s \colon H_s \to \mathcal{A}_s \mid s \in S)$ be a family of functions. There is a unique Σ-homomorphism ψ from $\mathrm{T}_\Sigma(H)$ to $(\mathcal{A}, \cdot^\mathcal{A})$ that *extends* φ, i.e., for each $s \in S$ and $h \in H_s$, it holds that $\varphi_s(h) = \psi_s(h)$. □

Proof. Let ψ, ψ' be Σ-homomorphisms from $\mathrm{T}_\Sigma(H)$ to $(\mathcal{A}, \cdot^\mathcal{A})$ that extend φ. We prove by structural induction on $\mathrm{T}_\Sigma(H)$ that $\psi_s(\xi) = \psi'_s(\xi)$ for each $s \in S$ and $\xi \in (\mathrm{T}_\Sigma(H))_s$.

Induction base: Let $s \in S$ and $h \in H_s$. Then $\psi_s(h) = \varphi_s(h) = \psi'_s(h)$.

Induction step: Let $k \in \mathbb{N}$, $s_1, \ldots, s_k \in S$, $\sigma \in \Sigma_{(s_1 \cdots s_k, s)}$, and $\xi_1 \in (\mathrm{T}_\Sigma(H))_{s_1}$, $\ldots$, $\xi_k \in (\mathrm{T}_\Sigma(H))_{s_k}$ such that $\xi = \sigma(\xi_1, \ldots, \xi_k)$ and, for each $i \in [k]$, we have $\psi_{s_i}(\xi_i) = \psi'_{s_i}(\xi_i)$. Then:

$$\begin{aligned}
\psi_s(\sigma(\xi_1, \ldots, \xi_k)) &= \psi_s(\sigma^{\mathrm{T}_\Sigma(H)}(\xi_1, \ldots, \xi_k)) \\
&= \sigma^\mathcal{A}(\psi_{s_1}(\xi_1), \ldots, \psi_{s_k}(\xi_k)) && (\psi_s \text{ homomorphism}) \\
&= \sigma^\mathcal{A}(\psi'_{s_1}(\xi_1), \ldots, \psi'_{s_k}(\xi_k)) && (k \text{ times induction hypothesis}) \\
&= \psi'_s(\sigma^{\mathrm{T}_\Sigma(H)}(\xi_1, \ldots, \xi_k)) && (\psi'_s \text{ homomorphism}) \\
&= \psi'_s(\sigma(\xi_1, \ldots, \xi_k))
\end{aligned}$$

■

We extend the running example by defining an algebra for evaluating the expression over truth values and integers represented by some term to its value.

Example 2.3.9 (Example 2.3.4 continued)**.** We define a Σ-algebra $(\mathcal{A}, \cdot^\mathcal{A})$ where $\mathcal{A}_\mathrm{b} = \{\top, \bot\}$, $\mathcal{A}_\mathrm{i} = \mathbb{N}$, and

$$\oplus^\mathcal{A}(n_1, n_2) = n_1 + n_2 \qquad \otimes^\mathcal{A}(n_1, n_2) = n_1 \cdot n_2$$

$$\circledcirc^\mathcal{A}(n_1, n_2) = \begin{cases} \top & \text{if } n_1 < n_2 \\ \bot & \text{otherwise} \end{cases} \qquad \mathrm{ite}^\mathcal{A}(b, n_1, n_2) = \begin{cases} n_1 & \text{if } b = \top \\ n_2 & \text{if } b = \bot \end{cases}$$

Consider the family of functions $\varphi = (\varphi_s \colon H_s \to \mathcal{A}_s \mid s \in S)$ such that $\varphi_s(h) = h$ for each $s \in S$ and $h \in H$. Then the application of the unique extension of φ to a Σ-homomorphism ψ from $\mathrm{T}_\Sigma(H)$ to $\mathcal{A}$ to our running example is as follows:

$$\begin{aligned}
&\psi_\mathrm{i}(\mathrm{ite}(\circledcirc(1,2), \oplus(3,4), \otimes(2,2))) \\
&= \mathrm{ite}^\mathcal{A}(\psi_\mathrm{b}(\circledcirc(1,2)), \psi_\mathrm{i}(\oplus(3,4)), \psi_\mathrm{i}(\otimes(2,2))) \\
&= \mathrm{ite}^\mathcal{A}(\circledcirc^\mathcal{A}(\psi_\mathrm{i}(1), \psi_\mathrm{i}(2)), \oplus^\mathcal{A}(\psi_\mathrm{i}(3), \psi_\mathrm{i}(4)), \otimes^\mathcal{A}(\psi_\mathrm{i}(2), \psi_\mathrm{i}(2))) \\
&= \mathrm{ite}^\mathcal{A}(\circledcirc^\mathcal{A}(\varphi_\mathrm{i}(1), \varphi_\mathrm{i}(2)), \varphi_\mathrm{i}(3) + \varphi_\mathrm{i}(4), \varphi_\mathrm{i}(2) \cdot \varphi_\mathrm{i}(2)) \\
&= \mathrm{ite}^\mathcal{A}(\circledcirc^\mathcal{A}(1,2), 3 + 4, 2 \cdot 2) \\
&= \mathrm{ite}^\mathcal{A}(\top, 7, 4) \\
&= 7
\end{aligned}$$

□

In the following we mostly consider sorted trees over an empty index set. We introduce a special notation for the unique Σ-homomorphism from the corresponding Σ-term algebra into some Σ-algebra $\mathcal{A}$.

Definition 2.3.10. Let S be a countable set, Σ an $S^* \times S$-sorted alphabet, and H an S-sorted set with $H_s = \emptyset$ for each $s \in S$. We denote the s-component of the unique Σ-homomorphism from $\mathrm{T}_\Sigma(H)$ to $(\mathcal{A}, \cdot^\mathcal{A})$ by $[\![\cdot]\!]_s^\mathcal{A}$, where we write $[\![t]\!]_s^\mathcal{A}$ instead of $[\![\cdot]\!]_s^\mathcal{A}(t)$. □

We also say that trees in T_Σ, i.e., the elements of the domain of the Σ-term algebra, are *interpreted in the algebra* $\mathcal{A}$. In particular, $[\![t]\!]_s^\mathcal{A}$ is called the *interpretation of* t *in* $\mathcal{A}$.

For our purposes it will be important to interpret terms over some alphabet Σ in multiple algebras that may require different sorted extensions of Σ. This is unproblematic as long as the sorted extensions of Σ agree on the number of arguments each symbol has.

Definition 2.3.11 (Compatible sorts and product algebras)**.** Let S and T be countable sets and Σ be an alphabet. Let $\mathrm{sort}_1 \colon \Sigma \to S^* \times S$ and $\mathrm{sort}_2 \colon \Sigma \to T^* \times T$, i.e., $(\Sigma, \mathrm{sort}_1)$ is an $S^* \times S$-sorted alphabet and $(\Sigma, \mathrm{sort}_2)$ is a $T^* \times T$-sorted alphabet. We say that sort_1 and sort_2 are *compatible* if, for each $\sigma \in \Sigma$, $u \in S^*$, $s \in S$, $v \in T^*$, and $t \in T$, we have that $(u, s) = \mathrm{sort}_1(\sigma)$ and $(v, t) = \mathrm{sort}_2(\sigma)$ implies $|u| = |v|$.

If sort_1 and sort_2 are compatible, we define $\mathrm{sort}_1 \times \mathrm{sort}_2 \colon \Sigma \to (S \times T)^* \times (S \times T)$ such that $(\mathrm{sort}_1 \times \mathrm{sort}_2)(\sigma) = ((s_1, t_1) \cdots (s_k, t_k), (s, t))$ for each $\sigma \in \Sigma$ where $\mathrm{sort}_1(\sigma) = (s_1 \dots s_k, s)$ and $\mathrm{sort}_2(\sigma) = (t_1 \dots t_k, t)$.

Let $(\mathcal{A}, \cdot^\mathcal{A})$ be a $(\Sigma, \mathrm{sort}_1)$-algebra and $(\mathcal{B}, \cdot^\mathcal{B})$ be a $(\Sigma, \mathrm{sort}_2)$-algebra where sort_1 and sort_2 are compatible. We define the *product algebra* $\mathcal{A} \times \mathcal{B}$ to be the $(\Sigma, \mathrm{sort}_1 \times \mathrm{sort}_2)$-algebra $(\mathcal{C}, \cdot^\mathcal{C})$ where $\mathcal{C}_{(s,t)} = \mathcal{A}_s \times \mathcal{B}_t$ for each $s \in S$ and $t \in T$ and, for each $\sigma \in \Sigma_{((s_1,t_1)\cdots(s_k,t_k),(s,t))}$ with $k \in \mathbb{N}$, $s, s_1, \dots, s_k \in S$, $t, t_1, \dots, t_k \in T$, $a_1 \in \mathcal{A}_{s_1}, \dots, a_k \in \mathcal{A}_{s_k}$, and $b_1 \in \mathcal{B}_{s_1}, \dots, b_k \in \mathcal{B}_{s_k}$ we have

$$\sigma^\mathcal{C}((a_1, b_1), \dots, (a_k, b_k)) = (\sigma^\mathcal{A}(a_1, \dots, a_k), \sigma^\mathcal{B}(b_1, \dots, b_k)) \ . \qquad \square$$

Observation 2.3.12. Since sort_1 and sort_2 may restrict in different ways which symbols can be combined to form a tree, we have for each $(s, t) \in S \times T$ that

$$\left(\mathrm{T}_{(\Sigma, \mathrm{sort}_1 \times \mathrm{sort}_2)}\right)_{(s,t)} \subseteq \left(\mathrm{T}_{(\Sigma, \mathrm{sort}_1)}\right)_s \cap \left(\mathrm{T}_{(\Sigma, \mathrm{sort}_2)}\right)_t \ . \qquad \square$$

2.3.2 Regular tree grammars

In this subsection we describe mechanisms to characterize particular subsets of the sets of all trees that can be formed over an alphabet Σ at hand. We are going to use so-called *regular tree grammars* for this purpose. Concerning the alphabet Σ we will initially ignore the sorts of the arguments of a symbol in Σ – it suffices to know how many arguments each symbol accepts.

Definition 2.3.13 (Ranked alphabet)**.** Let $S = \{\iota\}$ be a singleton and $(\Sigma, \mathrm{sort}_\Sigma)$ an $S^* \times S$-sorted alphabet. We also call $(\Sigma, \mathrm{sort}_\Sigma)$ *ranked alphabet* and equivalently

specify it by a tuple $(\Sigma, \mathrm{rk}_\Sigma)$ where $\mathrm{rk}_\Sigma \colon \Sigma \to \mathbb{N}$ is such that, for each $w \in S^*$ and $\sigma \in \Sigma_{(w,\iota)}$, we have $\mathrm{rk}(\sigma) = |w|$. We abbreviate $(\Sigma, \mathrm{rk}_\Sigma)$ by Σ and, for each $k \in \mathbb{N}$, denote by Σ_k the set $\{\sigma \in \Sigma \mid \mathrm{rk}_\Sigma(\sigma) = k\}$. □

In examples we may specify a ranked alphabet (Σ, rk) by writing $\Sigma = \{\sigma_1^{(k_1)}, \ldots, \sigma_n^{(k_n)}\}$ if $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and $\mathrm{rk}_\Sigma(\sigma_i) = k_i$ for each $i \in [n]$.

Definition 2.3.14. Let Σ be a ranked alphabet and N be an alphabet disjoint from Σ. We define the $N^* \times N$-sorted alphabet $\mathrm{R}[N, \Sigma]$, where for each $k \in \mathbb{N}$ and $B, B_1, \ldots, B_k \in N$, we have

$$\mathrm{R}[N, \Sigma]_{(B_1 \cdots B_k, B)} = \{B \to \sigma(B_1, \ldots, B_k) \mid \sigma \in \Sigma_k\} \ . \qquad \square$$

Next we define regular tree grammars, a device that can generate languages (i.e., sets) of trees. We restrict ourself to regular tree grammars in a particular normal form where each rule has exactly one terminal symbol[4] on its right-hand side. The reader may recognize that this definition is an obvious syntactic variant of tree automata with a single final state, however, the equivalence of regular tree grammars and tree-automata holds also in general (see Brainerd 1969; Gécseg and Steinby 1984, Theorem 2.3.6).

Definition 2.3.15. A *regular tree grammar* (RTG, Brainerd 1968; Brainerd 1969; Thatcher and Wright 1968) is a quadruple $G = (N, \Sigma, S, R)$ where

- N is an alphabet, whose elements are called *nonterminals*,
- Σ is a ranked alphabet, whose elements are called *terminals*,[5]
- $S \in N$, called *initial nonterminal*, and
- R is an $N^* \times N$-sorted alphabet where $R_s \subseteq \mathrm{R}[N, \Sigma]_s$ for each $s \in N^* \times N$.

We call the elements of R *rules* and the elements of $(\mathrm{T}_R)_S$ *derivation trees of* G. The *rule-projection algebra* $(\Pi, \cdot^\Pi)$ is an R-algebra where for each $B \in N$ we have $\Pi_B = \mathrm{T}_\Sigma$ and for each rule ϱ of form

$$B \to \sigma(B_1, \ldots, B_k) \tag{2.1}$$

it holds that $\varrho^\Pi(t_1, \ldots, t_k) = \sigma(t_1, \ldots, t_k)$ given $t_1, \ldots, t_k \in \mathrm{T}_\Sigma$. The *tree language of* G, denoted by $L(G)$, is $\{[\![d]\!]_S^\Pi \mid d \in (\mathrm{T}_R)_S\}$. For each $B \in N$, we define the *B-fragment* as the set $R_B = \bigcup_{w \in N^*} R_{(w,B)}$. □

[4] We follow the conventions of computer science and call every symbol that occurs in the tree a terminal symbol. In contrast, linguists tend to call only symbols occurring in leaf positions terminal symbols.

[5] We do *not* require that $N \cap \Sigma = \emptyset$ because there is no risk of confusing terminal and nonterminal symbols.

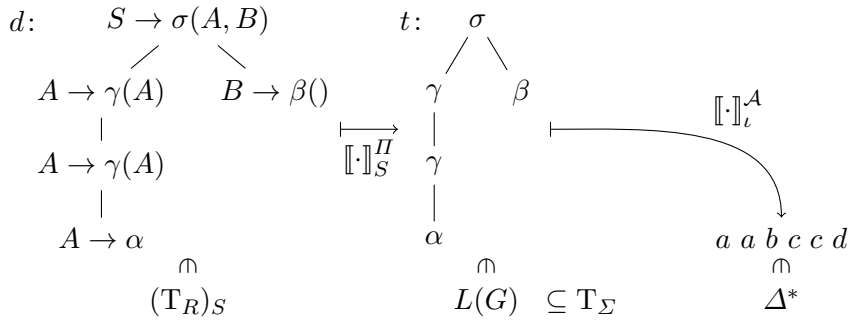


Figure 2.1: A derivation tree d, its projection to a tree t in $L(G) \subseteq \mathrm{T}_\Sigma$, and t's interpretation to a string in Δ^*.

Example 2.3.16. Let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ be a ranked alphabet. We define the RTG $G = (N, \Sigma, S, R)$ where $N = \{S, A, B\}$ and

$$R = \{(S \to \sigma(A, B))^{(AB,S)}, (A \to \gamma(A))^{(A,A)}, (A \to \alpha())^{(\varepsilon,A)}, (B \to \beta())^{(\varepsilon,B)}\} \ .$$

Figure 2.1 shows a derivation tree d in $(\mathrm{T}_R)_S$ and its projection to a tree t in T_Σ. □

An important property of regular tree languages, i.e., the tree languages generated by regular tree grammars, is that they are closed under intersection.

Definition 2.3.17. Let Σ be a ranked alphabet and let $G_1 = (N_1, \Sigma, S_1, R_1)$ and $G_2 = (N_2, \Sigma, S_2, R_2))$ be RTGs. We define the RTG $G_1 \times G_2 = (N_1 \times N_2, \Sigma, (S_1, S_2), R')$ where

$$\begin{aligned} R' = \{(A, B) \to \sigma((A_1, B_1), \ldots, (A_k, B_k)) \mid k \in \mathbb{N}, \sigma \in \Sigma_k, \\ A \to \sigma(A_1, \ldots, A_k) \text{ in } R_1, \\ B \to \sigma(B_1, \ldots, B_k) \text{ in } R_2\} \ . \end{aligned}$$ □

Theorem 2.3.18 (Gécseg and Steinby 1984, Theorem 2.4.2)**.** Let Σ be a ranked alphabet and G_1 and G_2 be RTGs over Σ. Then $L(G_1 \times G_2) = L(G_1) \cap L(G_2)$. □

2.3.3 Initial algebra semantics

Let S be a countable set and $\mathcal{A}$ be an S-indexed family. In order to characterize some subset U of $\mathcal{A}_s$ with $s \in S$, Goguen et al. (1977) proposed *initial algebra semantics*. To this end, an $S^* \times S$-sorted alphabet $(\Sigma, \mathrm{sort}_\Sigma)$ and a tree language $T \subseteq (\mathrm{T}_\Sigma)_s$ are considered such that $(\mathcal{A}, \cdot^\mathcal{A})$ is a Σ-algebra and $U = [\![T]\!]_s^\mathcal{A}$.

It comes natural to use an RTG to specify T:

Definition 2.3.19. Let S be a countable set and $(\Sigma, \mathrm{sort}_\Sigma)$ be an $S^* \times S$-sorted alphabet. Let G be an RTG $(N, (\Sigma, \mathrm{rk}), \bar{S}, R)$ where, for each $\sigma \in \Sigma$, we require

$\mathrm{rk}(\sigma) = |w|$ with $(w, s) = \mathrm{sort}_\Sigma(\sigma)$. G is called $(\Sigma, \mathrm{sort}_\Sigma)$-*compatible* if there is a function $\mathrm{sort}_N \colon N \to S$ satisfying, for each rule $A_0 \to \sigma(A_1, \ldots, A_k)$ in R, that

$$\mathrm{sort}_\Sigma(\sigma) = (\mathrm{sort}_N(A_1) \cdots \mathrm{sort}_N(A_k), \mathrm{sort}_N(A_0)) \ . \qquad \square$$

Remark. For each nonterminal $A \in N$ that is *useful*, i.e., that occurs in at least one rule in R, $\mathrm{sort}_N(A)$ is determined by sort_Σ. Thus, if each nonterminal of G is useful and sort_N exists, then sort_N is unique. Otherwise, we may choose an arbitrary candidate.

Following Koller and Kuhlmann (2011) we combine an RTG G and a fixed number algebras that interpret G's tree language to an *interpreted regular tree grammar*.

Definition 2.3.20 (Interpreted regular tree grammar)**.** Let $(\Sigma, \mathrm{rk}_\Sigma)$ be a ranked alphabet. An *interpreted regular tree grammar* (IRTG)[6] is a tuple $(G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ where

- $k > 0$,
- for each $i \in [k]$, $\mathcal{A}_i$ is a $(\Sigma, \mathrm{sort}_i)$-algebra, and
- $G = (N, (\Sigma, \mathrm{rk}_\Sigma), S, R)$ is an RTG that is $(\Sigma, \mathrm{sort}_i)$-compatible for each $i \in [k]$.

For each $i \in [k]$, we refer to the function sort_N that exists due to $(\Sigma, \mathrm{sort}_i)$-compatibility of G by sort_N^i. $\square$

IRTG can for instance be used to simulate context-free grammars (CFG) as the next example illustrates.

Example 2.3.21. Let Σ and $G = (N, \Sigma, S, R)$ be as in Example 2.3.16. Consider the set $S = \{\iota\}$, the function $\mathrm{sort} \colon \Sigma \to S$, the alphabet $\Delta = \{a, b, c, d\}$, and the (Σ, sort)-algebra $\mathcal{A} = (\Delta^*, \cdot^{\mathcal{A}})$ where

$$\begin{array}{lll}
\mathrm{sort}(\sigma) = (\iota\iota, \iota) & \sigma^{\mathcal{A}}(x, y) = x \cdot y & (S \to AB) \\
\mathrm{sort}(\gamma) = (\iota, \iota) & \gamma^{\mathcal{A}}(x) = a \cdot x \cdot c & (A \to aAc) \\
\mathrm{sort}(\alpha) = (\varepsilon, \iota) & \alpha^{\mathcal{A}} = b & (A \to b) \\
\mathrm{sort}(\beta) = (\varepsilon, \iota) & \beta^{\mathcal{A}} = d & (B \to d) \ .
\end{array}$$

Note that G is $(\Sigma, \mathrm{sort}_\Sigma)$-compatible, i.e., $\mathbb{G} = (G, \mathcal{A})$ is an IRTG. The interpretation of a tree $t \in L(G)$ in the algebra $\mathcal{A}$ to a string in Δ^* is depicted in Figure 2.1. The rules of an CFG equivalent to $\mathbb{G}$ are given in the last column of the above table . $\square$

[6] The definition of IRTG in this work differs from the one by Koller and Kuhlmann (2011); see Remark 2.3.25.

Abstractly, an IRTG can be used to describe languages of objects a which are elements of the Cartesian product of the domains of the algebras $\mathcal{A}_1$, ..., $\mathcal{A}_k$, respectively. We refer to the set $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ also as the *domain* of the IRTG at hand. A central problem associated to IRTG is the *parsing problem*.

Definition 2.3.22. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG and $a = (a_1, \ldots, a_k)$ in $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_k$. The *parsing problem for* $\mathbb{G}$ *and* a is to compute

$$\mathrm{parses}_{\mathbb{G}}(a) = \{t \in L(G) \mid \forall j \in [k] : [\![t]\!]^{\mathcal{A}_j}_{\mathrm{sort}^j_N(S)} = a_j\} \ . \qquad \square$$

Of special interest for this thesis is solving the parsing problem for IRTGs where the algebras are *regularly decomposable*:

Definition 2.3.23 (Koller and Kuhlmann 2011, Def. 3)**.** Let S be a set of sorts and Σ be an $S^* \times S$-sorted alphabet. A Σ-algebra $\mathcal{A}$ is called *regularly decomposable* if there is a computable function D which maps every object $a \in \mathcal{A}_s$ (where $s \in S$) to a regular tree grammar $D(a)$ such that

$$L(D(a)) = \{t \in (\mathrm{T}_\Sigma)_s \mid [\![t]\!]^{\mathcal{A}}_s = a\}. \qquad \square$$

In fact, many algebras that are of interest for natural language processing are regularly decomposable, e.g., the algebras that can be used to model context-free grammars and linear context-free rewriting systems (see Theorem 6.1.1). For illustration, we also give an example for an algebra that is *not* regularly decomposable:

Example 2.3.24. Let $S = \{\iota\}$, $\Sigma = \{\mathrm{inc}^{(\iota,\iota)}, \mathrm{dec}^{(\iota,\iota)}, \mathrm{zero}^{(\varepsilon,\iota)}\}$ be an S-sorted alphabet, and $\mathcal{A} = (\mathbb{Z}, \cdot^{\mathcal{A}})$ be a Σ-algebra with $\mathrm{inc}^{\mathcal{A}}(n) = n + 1$, $\mathrm{dec}^{\mathcal{A}}(n) = n - 1$, and $\mathrm{zero}^{\mathcal{A}}() = 0$. If $\mathcal{A}$ was regularly decomposable, then for each $n \in \mathbb{Z}$ there would be an RTG $D(n)$ with

$$L(D(n)) = \{t \in \mathrm{T}_\Sigma \mid |\{p \in \mathrm{pos}(t) \mid t(p) = \mathrm{inc}\}| - |\{p \in \mathrm{pos}(t) \mid t(p) = \mathrm{dec}\}| = n\} \ .$$

For each number $m \in \mathbb{N}$, the tree $\mathrm{inc}^{m+n}(\mathrm{dec}^m(\mathrm{zero}))$ would be in $L(D(n))$. Let the cardinality of $D(n)$'s set of nonterminals be q. Due to pumping arguments we have that for some $1 < i \leq q$ and $m \gg q$ also $\mathrm{inc}^{m+i+n}(\mathrm{dec}^m(\mathrm{zero}))$ is in $L(D(n))$, a contradiction. $\square$

If, for each $j \in [k]$, $\mathcal{A}_j$ is regularly decomposable, then $\mathrm{parses}_{\mathbb{G}}(a)$ is equal to

$$\bigcap_{j \in [k]} L(D(a_j)) \cap L(G) \ .$$

In the remainder of this thesis we consider only such algebras. Since the languages of RTGs are closed under intersection (cf. Theorem 2.3.18), we can construct, for each $j \in [k]$, an RTG G_{a_j}, called *chart of* a_j, with $L(G_{a_j}) = L(D(a_j)) \cap L(G)$. Moreover, we

can construct the RTG G_a, called *chart of* a, with $L(G_a) = \bigcap_j L(G_{a_j}) = \mathrm{parses}_{\mathbb{G}}(a)$.

Let $I \subseteq [k]$. We define $\mathcal{A}_I = \times_{i \in I} \mathcal{A}_i$ where we assume that the elements of I are in a fixed order $i_1, \ldots, i_m$, e.g., ascending. For each $t \in \bigcap_{i \in I} (\mathrm{T}_\Sigma)_{\mathrm{sort}_N^i(S)}$, we define

$$[\![t]\!]^{\mathcal{A}_I} = ([\![t]\!]^{\mathcal{A}_{i_1}}_{\mathrm{sort}_N^{i_1}(S)}, \ldots, [\![t]\!]^{\mathcal{A}_{i_m}}_{\mathrm{sort}_N^{i_m}(S)}) \ .$$

Remark 2.3.25. Originally, Koller and Kuhlmann (2011) define IRTG over non-sorted Σ-algebras. Moreover, they accompany each algebra $\mathcal{A}_j$ in an IRTG with a tree-homomorphism $h_j \colon \mathrm{T}_\Sigma \to \mathrm{T}_{\Gamma_j}$ and make $\mathcal{A}_j$ a Γ_j-algebra. Here, Γ_j is a ranked alphabet of atomic operations specific to the algebra $\mathcal{A}_j$. For instance, an algebra for context-free grammars over some alphabet Δ would use the operator symbols $\Gamma = \{\cdot^{(2)}\} \cup \{\delta^{(0)} \mid \delta \in \Delta\} \cup \{\varepsilon^{(0)}\}$ where $\cdot$ is interpreted as the concatenation function, each symbol $\delta \in \Delta$ is interpreted as itself, and ε is interpreted as the empty word.

A *tree homomorphism* is a function h from T_Σ to T_Γ that is obtained by extending a function $h' \colon \Sigma \to \mathrm{T}_\Gamma(Y_k)$ such that $h(\sigma(t_1, \ldots, t_k)) = h'(\sigma)[y_i / h(t_i) \mid i \in [k]]$. For Example 2.3.21 we could define h as follows:

$$\begin{aligned} h'(\sigma) &= \cdot(y_1, y_2) \\ h'(\gamma) &= \cdot(\cdot(a, \cdot(y_1, c))) \\ h'(\alpha) &= b \\ h'(\beta) &= d \end{aligned}$$

Defining IRTG with such an additional homomorphism h has the advantage that for each algebra only a limited set of atomic operations needs to be specified. Complex operations can then be constructed as terms over the atomic operations. This also allows for the development of generic binarization techniques as in Büchse, Koller, and Vogler (2013): Σ, G, and h are altered to Σ', G', and h', respectively, such that each symbol in Σ has at most rank 2. However, the algebra $\mathcal{A}$ does not change and $\{[\![h(t)]\!]^{\mathcal{A}} \mid t \in L(G)\} = \{[\![h'(t)]\!]^{\mathcal{A}} \mid t \in L(G')\}$. This is not possible if the operations of $\mathcal{A}$ have more than 2 arguments, which is more likely to be the case for complex operations.

For the techniques discussed in this thesis we do not need to alter the alphabet Σ and therefore drop the intermediate homomorphisms to keep the presentation more concise. Still our model is equally expressive: for each tree homomorphism $h \colon \mathrm{T}_\Sigma \to \mathrm{T}_\Gamma$ and Γ-algebra $\mathcal{A} = (\mathcal{A}, \cdot^{\mathcal{A}})$ there is a Σ-algebra $\mathcal{A}' = (\mathcal{A}', \cdot^{\mathcal{A}'})$ such that $[\![\cdot]\!]^{\mathcal{A}} \circ h = [\![\cdot]\!]^{\mathcal{A}'}$. We let the carrier set of $\mathcal{A}'$ be the same as the one of $\mathcal{A}$. The operations of the algebra $\mathcal{A}'$ can be constructed as follows: Let $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $h'(\sigma) = \xi$ in $\mathrm{T}_\Gamma(Y_k)$. We set $\sigma^{\mathcal{A}'}(a_1, \ldots, a_k) = g'(\xi)$ where $g' \colon \mathrm{T}_\Gamma(Y_k) \to \mathcal{A}$ is the unique Γ-homomorphism that extends $g \colon Y_k \to \mathcal{A}$ with $g(y_i) = a_i$ for each $i \in [k]$. □

2.4 Unranked trees and hedges

Unranked trees and their recognizers have been intensively studied in the theoretical computer science community (Thatcher 1967; Libkin 2006). Unranked trees are trees where nodes with a particular label may have varying numbers of children. They occur, for instance, as the parse trees of context-free grammars, in XML documents, and in the syntax trees used in linguistics[7]. A sequence of child nodes in an unranked tree is often called a *hedge*. The term was supposedly coined by Bruno Courcelle and is also used to refer to sequences of trees at the top level. For a survey on unranked hedges and their recognizers, we refer to the technical report by Brüggemann, Murata, and Wood (2001).

In this thesis, unranked hedges are used to represent syntactic hierarchies. This is a deviation of earlier work with hybrid grammars (Nederhof and Vogler 2014; Gebhardt, Nederhof, and Vogler 2017), where so-called *sequence terms* (see Def. 3.1 in Fischer 1968 and Sec. 2.2 in Seki and Kato 2008) are used instead. Sequence terms can be thought of as hedges over ranked symbols, where each symbol has a fixed number of subhedges. In practice, Nederhof and Vogler (2014) and Gebhardt, Nederhof, and Vogler (2017) only use symbols of rank 0 or 1 to represent syntactic hierarchy, i.e., the expressiveness of sequence terms is rarely exhausted. In fact, the only advantage of this restriction is that certain symbols are forbidden to have any children. However, this can usually also be encoded in the state behavior of devices that are employed to recognize the sequence term (or unranked hedge). Still, as we later use an algebraic definition of simple definite clause programs (a device to define hedge languages), we will need ranked second-order variables in the otherwise unranked hedges.

Definition 2.4.1. Let Σ be an alphabet, I a set, and X be a ranked alphabet (*second-order variables*) such that Σ, I, and X are pairwise disjoint[8]. The set of *unranked trees over Σ and X indexed by I*, denoted by $\mathrm{U}_\Sigma(I, X)$ is the smallest set $V \subseteq (\Sigma \cup I \cup X \cup \{\,(\,,\,)\,\})^*$ such that

- $I \subseteq V$,
- for each $\sigma \in \Sigma$ and $w \in V^*$, we have $\sigma(w) \in V$, and
- $X(V) \subseteq V$.

We call $\mathrm{U}_\Sigma(I, X)^*$ the set of *unranked hedges over Σ and X indexed by I* and may write $\mathrm{U}^*_\Sigma(I, X)$ instead of $\mathrm{U}_\Sigma(I, X)^*$. We use the following abbreviations:

- U_Σ for $\mathrm{U}_\Sigma(\emptyset, \emptyset)$,
- U^*_Σ for $\mathrm{U}^*_\Sigma(\emptyset, \emptyset)$,

[7] Some syntactic theories allow only for at most binary nodes and sometimes the order of the children is not relevant (cf. Müller 2016).

[8] W.l.o.g. we may assume that the symbols "(", ")", and "," are not elements of Σ, I, or X.

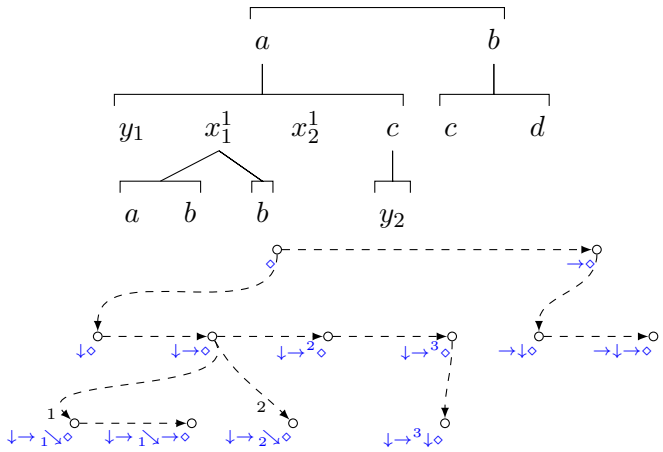


Figure 2.2: An unranked hedge (top) and its positions (bottom).

- $\mathrm{U}_\Sigma(I)$ for $\mathrm{U}_\Sigma(I, \emptyset)$, and
- $\mathrm{U}^*_\Sigma(I)$ for $\mathrm{U}^*_\Sigma(I, \emptyset)$.

An unranked tree of the form $\sigma()$ for $\sigma \in \Sigma$ is also denoted by σ. □

Definition 2.4.2. Let $v \in \mathrm{U}^*_\Sigma(I, X)$ such that there are $n \in \mathbb{N}$ and $\xi_1, \ldots, \xi_n \in \mathrm{U}_\Sigma(I, X)$ with $v = \xi_1 \cdots \xi_n$. We define the *length of* v, denoted by $\mathrm{len}(v)$, to be the number n. □

Example 2.4.3. Consider the alphabet $\Sigma = \{a, b, c, d\}$, the set $Y_2 = \{y_1, y_2\}$, and $X = \{x^1_1, x^1_2\}$ with $\mathrm{rk}_X(x^1_1) = 2$ and $\mathrm{rk}_X(x^1_2) = 0$. The hedge

$$\xi = a(y_1\ x^1_1(a\ b, b)\ x^1_2\ c(y_2))\ b(c\ d)$$

is in $\mathrm{U}^*_\Sigma(Y, X)$ and graphically depicted in Figure 2.2: each symbol is represented as a node with its argument hedge (or hedges) arranged below. Multiple unranked trees are subsumed to a hedge by a square bracket above. □

Next, we define positions for unranked trees. We deviate from the usual Gorn addresses (cf. Gorn 1967) because later, we will substitute variables occurring in some tree or hedge by hedges of varying lengths. Due to this substitution, the position of a symbol right of a variable can get shifted arbitrarily, which would require cumbersome additions or subtractions to the numbers occurring in a Gorn address. Instead, we use addresses composed of unary movements in a tree (or hedge), where "$\rightarrow$" means "move to the next tree in the hedge", "$\downarrow$" means "move to the subhedge of the tree at the current position", "$\overset{i}{\searrow}$" means "go to the i-th argument" of a second-order variable, and "$\diamond$" means "inspect the subtree or symbol at the current position".

Definition 2.4.4. Let $\xi \in \mathrm{U}_\Sigma(I,X)$ and let $v \in \mathrm{U}^*_\Sigma(I,X)$. The set of *position markers* is $\mathcal{PM} = \{\diamond, \rightarrow, \downarrow\} \cup \{{}_i\searrow \mid i > 0\}$. The sets of *positions of* ξ and *positions of* v, denoted by $\mathrm{pos}(\xi)$ and $\mathrm{pos}^*(v)$, respectively, are subsets of $\mathcal{PM}^*$ defined recursively such that

$$\mathrm{pos}(\xi) = \begin{cases} \{\diamond\} & \text{if } \xi \in I \\ \{\diamond\} \cup (\{\downarrow\} \cdot \mathrm{pos}^*(w)) & \text{if } \xi = \sigma(w) \text{ for } \sigma \in \Sigma \\ & \text{ and } w \in \mathrm{U}^*_\Sigma(I,X) \\ \{\diamond\} \cup \{{}_i\searrow p \mid i \in [k], p \in \mathrm{pos}(\xi_i)\} & \text{if } \xi = x(\xi_1,\ldots,\xi_k) \text{ for } k \in \mathbb{N}, x \in X_k, \\ & \text{ and } \xi_1,\ldots,\xi_k \in \mathrm{U}^*_\Delta(I,X) \end{cases}$$

and, if $v = \xi_1 \cdots \xi_n$ with $n \in \mathbb{N}$ and $\xi_1,\ldots,\xi_n \in \mathrm{U}_\Sigma(I,X)$, then

$$\mathrm{pos}^*(v) = \{\rightarrow^{i-1} \cdot p \mid i \in [n], p \in \mathrm{pos}(\xi_i)\} \ . \qquad \square$$

Example 2.4.5 (Example 2.4.3 cont'd)**.** Figure 2.2 shows the positions of the unranked edge from Example 2.4.3. $\square$

In the following we extend our notation for unranked trees and hedges by definitions for the concepts "label at position", "subtree", "parent", "children", and "context", which are well-known for ranked trees (i.e., trees where the node label determines the number of successors).

Definition 2.4.6. Let $v \in \mathrm{U}^*_\Sigma(I,X) \cup \mathrm{U}_\Sigma(I,X)$ and $p \in \mathrm{pos}(v)$. The *label at position* p *of* v, denoted by $v(p)$, is defined recursively such that if $v \in \mathrm{U}^*_\Sigma(I,X)$, with $v = \xi_1 \cdots \xi_n$ for some $n \in \mathbb{N}$ and $\xi_1,\ldots,\xi_n \in \mathrm{U}_\Sigma(I,X)$, and $p = \rightarrow^{i-1} p'$ with $i \in [n]$ and $p' \in \mathrm{pos}(\xi_i)$, then $v(p) = \xi_i(p')$. Otherwise, if $v \in \mathrm{U}_\Sigma(I,X)$, then

$$v(p) = \begin{cases} v & \text{if } p = \diamond \text{ and } v \in I \\ \sigma & \text{if } p = \diamond \text{ and } v = \sigma(w) \text{ for } \sigma \in \Sigma \text{ and } w \in \mathrm{U}^*_\Sigma(I,X) \\ x & \text{if } p = \diamond \text{ and } v = x(w_1,\ldots,w_k) \text{ for } k \in \mathbb{N}, x \in X_k, \\ & \text{ and } w_1,\ldots,w_k \in \mathrm{U}^*_\Delta(I,X) \\ w(p') & \text{if } p = \downarrow p' \text{ and } v = \sigma(w) \text{ for } \sigma \in \Sigma, w \in \mathrm{U}^*_\Sigma(I), \text{ and } p' \in \mathrm{pos}(w) \\ w_i(p') & \text{if } p = {}_i\searrow p' \text{ and } \xi = x(w_1,\ldots,w_k) \text{ for } k \in \mathbb{N}, x \in X_k, \\ & \qquad w_1,\ldots,w_k \in \mathrm{U}^*_\Delta(I,X), \text{ and } p' \in \mathrm{pos}(w_i) \ . \end{cases}$$

For each set $Z \subseteq \Sigma \cup I \cup X$, we let $\mathrm{pos}_Z(v) = \{p \in \mathrm{pos}(v) \mid v(p) \in Z\}$. $\square$

Definition 2.4.7. Let $v \in \mathrm{U}^*_\Sigma(I,X)$ and $p \in \mathrm{pos}^*(v)$. In particular, let $v = \xi_1 \cdots \xi_n$ with $n = \mathrm{len}(v)$ and $\xi_1,\ldots,\xi_n \in \mathrm{U}_\Sigma(I,X)$. We define the *subtree of* v *at* p, denoted

by $v|_p$, recursively as follows:

$$v|_p = \begin{cases} \xi_i & \text{if } p = \to^{i-1}\diamond \text{ with } i \in [n] \\ w|_{p'} & \text{if } p = \to^{i-1}\downarrow p' \text{ and } \xi_i = \sigma(w) \\ & \quad \text{with } i \in [n], p' \in \mathrm{pos}(w), \sigma \in \Sigma, w \in \mathrm{U}^*_\Sigma(I) \\ w_j(p') & \text{if } p = \to^{i-1} {}_j\!\searrow p' \text{ and } \xi_i = x(w_1, \ldots, w_k) \text{ with } i \in [n], k \in \mathbb{N}, \\ & \quad j \in [k], x \in X_k, w_1, \ldots, w_k \in \mathrm{U}^*_\Sigma(I, X), \text{ and } p' \in \mathrm{pos}(w_j) \end{cases} . \square$$

Definition 2.4.8. Let $\xi \in \mathrm{U}_\Sigma(I, X)$ and let $v \in \mathrm{U}^*_\Sigma(I, X)$. We define the function

$$\mathrm{parent}\colon \left(\mathcal{PM}^* \cdot (\{\downarrow\} \cup \{{}_j\!\searrow \mid j > 0\}) \cdot \{\to\}^* \cdot \{\diamond\}\right) \to \mathcal{PM}^*$$

such that, for each $p \in \mathcal{PM}^*$, $q \in \{\downarrow\} \cup \{{}_j\!\searrow \mid j > 0\}$, and $i \in \mathbb{N}$, we have

$$\mathrm{parent}(p \cdot q \cdot \to^i \cdot \diamond) = p\,\diamond \quad .$$

We define the function

$$\mathrm{children}^\xi\colon \mathrm{pos}(\xi) \to \mathrm{pos}(\xi)^*$$

such that, for each $p \in \mathcal{PM}^*$, if $p\diamond \in \mathrm{pos}_{\Sigma \cup U}(\xi)$, then we set

$$\mathrm{children}^\xi(p\diamond) = (p \downarrow \to^0 \diamond)(p \downarrow \to^1 \diamond) \cdots (p \downarrow \to^{n-1} \diamond)$$

where $n \in \mathbb{N}$ is the smallest number such that $(p \downarrow \to^n \diamond) \notin \mathrm{pos}(\xi)$. We denote n also by $\star_p$. Alternatively, if $p \in \mathrm{pos}_X(\xi)$ where $\mathrm{rk}_X(p(\xi)) = k$, then we set

$$\begin{aligned} \mathrm{children}^\xi(p\diamond) = {} & (p\,{}_1\!\searrow \to^0 \diamond) \cdots (p\,{}_1\!\searrow \to^{n_1 - 1} \diamond) \\ & \cdots \\ & (p\,{}_k\!\searrow \to^0 \diamond) \cdots (p\,{}_k\!\searrow \to^{n_k - 1} \diamond) \end{aligned} ,$$

where, for each $i \in [k]$, $n_i \in \mathbb{N}$ is the smallest number such that $(p\,{}_i\!\searrow \to^{n_i} \diamond) \notin \mathrm{pos}(\xi)$.

We define the total order $\prec$ on $\mathcal{PM}$ such that $\diamond \prec \downarrow \prec {}_1\!\searrow \prec {}_2\!\searrow \prec \cdots \prec \to$. $\square$

Remark 2.4.9. If $v \in \mathrm{U}^*_\Sigma(I, X)$ is of length 1, i.e., there is $\xi \in \mathrm{U}_\Sigma(I)$ with $v = \xi$, then we have that $\mathrm{pos}^*(v) = \mathrm{pos}(\xi)$. Therefore, we write pos instead of pos^* in the following.

Traversing over the positions of a hedge in the lexicographic order induced by $\prec$ is equivalent to a *pre-order* traversal, i.e., each position is visited before its children and the "sibling positions" to its right. $\square$

Definition 2.4.10. Let $I = Y_k$ for some $k \in \mathbb{N}$, X be a ranked alphabet, and let $v \in \mathrm{U}^*_\Sigma(Y_k, X)$ be such that each $y_i \in Y_k$ occurs exactly once in v. For each $i \in [k]$ denote the position where y_i occurs in v by p_i. If for each $i, i' \in [k]$ with $i < i'$ we have that $p_i \preceq_{\mathrm{lex}} p_{i'}$, then v is called *Y_k-context for Δ and X*. The *set of all Y_k-contexts for Δ and X which are in* $\mathrm{U}_\Sigma(Y_k, X)$ *(in* $\mathrm{U}^*_\Sigma(Y_k, X)$*)* is denoted by $\mathrm{C}_\Delta(Y_k, X)$ (by $\mathrm{C}^*_\Delta(Y_k, X)$). As before we may use the notation $\mathrm{C}_\Sigma(Y_k)$ and $\mathrm{C}^*_\Sigma(Y_k)$ if X is the empty set. $\square$

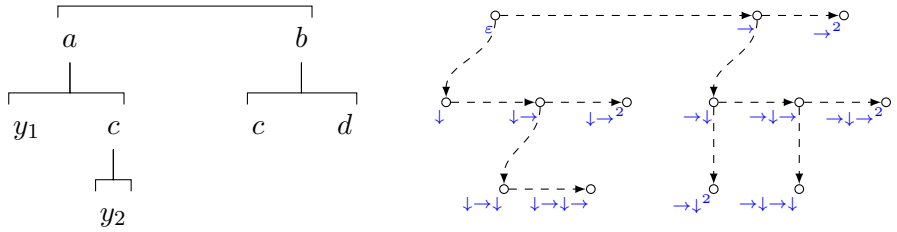


Figure 2.3: An unranked hedge and its span positions.

In order to refer to a subhedge (which we also call *span*) of some hedge, we define span positions. Note that we need and, thus, define this concept only for hedges without 2nd-order variables.

Definition 2.4.11. Let $\xi \in \mathrm{U}^*_\Delta(Y)$. The set of *span positions of* ξ, denoted by $\mathrm{spos}(\xi)$, is a subset of $\{\rightarrow, \downarrow\}^*$ defined recursively such that, if $\xi = \xi_1 \cdots \xi_n$ for $n \in \mathbb{N}$ and $\xi_1, \ldots, \xi_n \in \mathrm{U}_\Delta(Y)$, then

$$\begin{aligned}\mathrm{spos}(\xi) = &\{\rightarrow^i \mid 0 \leq i \leq n\} \\ &\cup \{\rightarrow^{i-1}\downarrow w \mid i \in [n], \xi_i = \delta(\xi') \text{ with } \delta \in \Delta, \xi' \in \mathrm{U}^*_\Delta(Y), w \in \mathrm{spos}(\xi')\} \ .\end{aligned}$$

Let $w \in \mathrm{spos}(\xi)$ and $i \in \mathbb{N}$ such that $w \rightarrow^i \in \mathrm{spos}(\xi)$. We call $(w, \rightarrow^i)$ a *span position pair for* ξ. We define the $(w, \rightarrow^i)$*-span of* ξ, denoted by $\xi|_w^{\rightarrow^i}$, recursively, such that if $\xi = \xi_1 \cdots \xi_n$ for $n \in \mathbb{N}$ and $\xi_1, \ldots, \xi_n \in \mathrm{U}_\Delta(Y)$, then

$$\xi|_w^{\rightarrow^i} = \begin{cases} \xi_{j+1} \cdots \xi_{j+i} & \text{if } w = \rightarrow^j \text{ with } j \in [n]_0 \\ \xi'|_{w'}^{\rightarrow^i} & \text{if } w = \rightarrow^{j-1}\downarrow w' \text{ with } j \in [n], \xi_j = \delta(\xi'), w' \in \mathrm{spos}(\xi') \ . \end{cases}$$

We say that $(w, \rightarrow^i)$ is *empty* if $i = 0$. □

Example 2.4.12. An example for the span positions of some hedge is given in Figure 2.3. We observe that each position of the hedge is embraced by two span positions: one to its left and one to its right. □

Definition 2.4.13. Let $(w, \rightarrow^i)$ and $(u, \rightarrow^j)$ be span position pairs for ξ.

(a) We say that $(w, \rightarrow^i)$ and $(u, \rightarrow^j)$ are *crossing* if $u = w \rightarrow^k$ for some $k \in [i-1]$ and $i < k + j$ or, symmetrically, if $w = u \rightarrow^k$ for some $k \in [j-1]$ and $j < k + i$.

(b) We say that $(w, \rightarrow^i)$ *encompasses* $(u, \rightarrow^j)$ if $u = w \rightarrow^k$ such that $0 \leq k \leq k + j \leq i$ and $(0 < k < i$ or $j > 0)$.

(c) We say that $(w, \rightarrow^i)$ and $(u, \rightarrow^j)$ are *overlapping*, if $(w, \rightarrow^i)$ and $(u, \rightarrow^j)$ are crossing, $(w, \rightarrow^i)$ *encompasses* $(u, \rightarrow^j)$, or $(u, \rightarrow^j)$ *encompasses* $(w, \rightarrow^i)$.

(d) We say that $(w, \rightarrow^i)$ *is below* $(u, \rightarrow^j)$, denoted by $(w, \rightarrow^i) \sqsubseteq (u, \rightarrow^j)$, if

- either $w = u \rightarrow^k$ for some $k \in [i]_0$ and $k + i \leq j$ or
- $w = u \rightarrow^k v$ for some $k \in [i-1]_0$ and $v \in \{\downarrow\} \cdot \{\rightarrow, \downarrow\}^*$.

$(w, \rightarrow^i)$ *is strictly below* $(u, \rightarrow^j)$, denoted by $(w, \rightarrow^i) \sqsubset (u, \rightarrow^j)$, if

- either $(u, \rightarrow^j)$ *encompasses* $(w, \rightarrow^i)$ or
- $w = u \rightarrow^k v$ for some $k \in [i-1]_0$ and $v \in \{\downarrow\} \cdot \{\rightarrow, \downarrow\}^*$.

(e) We say that $(w, \rightarrow^i)$ and $(u, \rightarrow^j)$ are *in parallel*, if they are not overlapping and neither $(w, \rightarrow^i) \sqsubset (u, \rightarrow^j)$ nor $(u, \rightarrow^j) \sqsubset (w, \rightarrow^i)$.

The relations *non-crossing*, *non-overlapping*, and *above* are defined analogously. □

Example 2.4.14.

- The span position pairs $(\downarrow, \rightarrow^2)$ and $(\downarrow\rightarrow, \rightarrow^2)$ are crossing but $(\downarrow, \rightarrow^2)$ and $(\downarrow\rightarrow^2, \rightarrow^2)$ are non-crossing.
- $(\downarrow, \rightarrow^2)$ encompasses each of the span position pairs $(\downarrow, \rightarrow^j)$, $j \in [2]$, and $(\downarrow\rightarrow, \rightarrow^j)$, $j \in \{0, 1\}$, but not $(\downarrow, \rightarrow^0)$ and $(\downarrow\rightarrow^2, \rightarrow^0)$.
- Each span position pair is below itself.
- $(\downarrow, \rightarrow^0) \sqsubseteq (\downarrow, \rightarrow^1)$ and $(\downarrow\rightarrow, \rightarrow^0) \sqsubseteq (\downarrow, \rightarrow^1)$ but *not* $(\downarrow, \rightarrow^0) \sqsubset (\downarrow, \rightarrow^1)$ and $(\downarrow\rightarrow, \rightarrow^0) \sqsubset (\downarrow, \rightarrow^1)$.
- $(w, \rightarrow^i) \sqsubset (u, \rightarrow^j)$ implies $(w, \rightarrow^i) \sqsubseteq (u, \rightarrow^j)$. □

2.5 Hybrid trees

In order to represent parse trees which are potentially discontinuous, we consider a data structure called *hybrid tree* (Nederhof and Vogler 2014). Originally, a hybrid tree is defined as an s-term (a data structure similar to hedges) and a linear order over a subset of the positions of the s-term. By considering only a subset of the nodes, this notion generalizes the *totally ordered trees* by Kuhlmann and Niehren (2008). If a discontinuous parse tree, as the one in Figure 1.2, shall be modeled as a hybrid tree, then exactly the leaf nodes are ordered. The linear order is chosen to match the order in which the words occur in the sentence.

We use a slightly more expressive variant of hybrid trees in this thesis. Our hybrid tree is a triple consisting of a string s, a hedge ξ, and an injective function α that maps each string position to a node of the tree. To model a discontinuous parse tree, we choose α such that each position in the sentence (represented by s) is mapped to some leaf node of ξ (which represents the syntactic hierarchy). Having an explicit representation of s and α fits the algebraic approach that we pursue. Also it gives us

the flexibility to align positions in s to positions in ξ with different labels, which is relevant for the implementation.

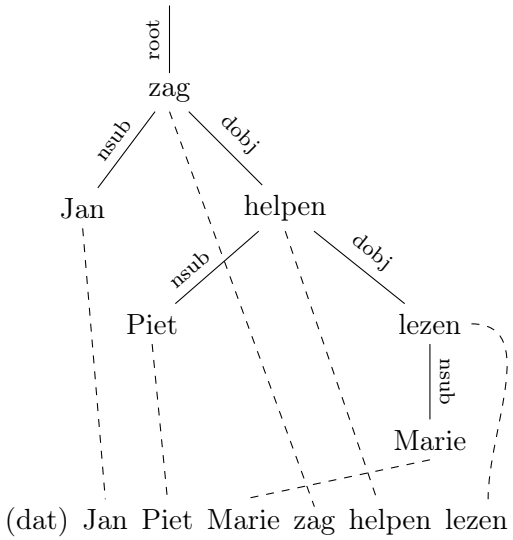


Figure 2.4: A non-projective dependency structure for a part of a Dutch sentence.

Next to the kind of parse trees considered so far, which are called *constituent trees* or *phrase structure trees*, there exists also another class of parse trees. These so-called *dependency trees* stem from a different tradition of syntactic theory (Tesnière 1959; Mel'čuk 1988). An example is given in Figure 2.4: the tree has exactly one node for each word of the sentence. The predicate of the sentence is usually the root of the tree. The children of each node are all other words that are arguments or adjuncts. Also here we can have a phenomenon that is related to discontinuity: if a node of the tree does not cover a continuous interval of sentence, then we call the tree *non-projective*. The approach to parsing that we develop is in principle applicable to both kinds of syntactic representation and capable to handle discontinuity and non-projectivity.

The next definition formalizes hybrid trees.

Definition 2.5.1. Let Σ and Δ be alphabets. A *hybrid tree* is a triple $h = (s, \xi, \alpha)$ where $\xi \in \mathrm{U}^*_\Sigma$, $s \in \Delta^*$, and $\alpha\colon [|s|] \to \mathrm{pos}(\xi)$ is injective. We call h *constituent tree* (or *phrase structure tree*), if $\{\alpha(i) \mid i \in [|s|]\} = \{p \in \mathrm{pos}(\xi) \mid \star_p = 0\}$, i.e., α is a bijection between the sentence positions and the leaf positions of ξ. We call h *dependency tree*, if $|s| = |\mathrm{pos}(\xi)|$. □

For some constructions, we need to refer to the sentence positions that are covered by a particular node inside a hybrid tree. This also facilitates a formal definition of the notions of discontinuity and non-projectivity.

Definition 2.5.2. Let $h = (s, \xi, \alpha)$ be a hybrid tree. We define $\text{cover}^h\colon \text{pos}(\xi) \to \mathfrak{P}([|s|])$ such that

$$\text{cover}^h(p\diamond) = \{i \in [|s|] \mid \alpha(i) \in \{p\} \cdot \{\diamond, \downarrow\} \cdot \{\rightarrow, \downarrow, \diamond\}^*\} \ .$$

We call h *projective*, if for each $p \in \text{pos}(\xi)$ there are $0 \leq i \leq j \leq |s|$ such that $\text{cover}^h(p) = \{i+1, i+2, \ldots, j\}$. Otherwise h is called *non-projective*. □

For constituent trees instead of projective and non-projective, the notions *continuous* and *discontinuous*, respectively, are used.

Example 2.5.3. We represent the discontinuous constituent tree from Figure 1.1 and the non-projective dependency structure from Figure 2.4 as hybrid trees h_1 and h_2, respectively. We set Δ to the set of words of the respective language. For Σ, we use the set of syntactic categories, i.e., S, NP, PRP, etc., in case of h_1 and the set of Dutch words ins case of h_2. Then:

- $h_1 = (s_1, \zeta_1, \alpha_1)$, where

$$\begin{aligned} s_1 &= \text{What shall I do} \ , \\ \zeta_1 &= \text{S(NP(PRP(I())) VP(MD(shall()) VC(WP(What()) VB(do()))))} \\ \alpha_1(1) &= (\downarrow\rightarrow)^2\downarrow^2\diamond \ , \quad \alpha_1(2) = \downarrow\rightarrow\downarrow^2\diamond \ , \quad \alpha_1(3) = \downarrow^3\diamond \ , \quad \text{and} \\ \alpha_1(4) &= (\downarrow\rightarrow)^3\downarrow\diamond \ . \end{aligned}$$

- $h_2 = (s_2, \zeta_2, \alpha_2)$, where

$$\begin{aligned} s_2 &= \text{Jan Piet Marie zag helpen lezen} \ , \\ \zeta_2 &= \text{zag(Jan() helpen(Piet() lezen(Marie())))} \ , \\ \alpha_2(1) &= \downarrow\diamond \ , \quad \alpha_2(2) = \downarrow\rightarrow\downarrow\diamond \ , \quad \alpha_2(3) = (\downarrow\rightarrow)^2\downarrow\diamond \ , \\ \alpha_2(4) &= \diamond \ , \quad \alpha_2(5) = \downarrow\rightarrow\diamond \ , \quad \text{and} \quad \alpha_2(6) = (\downarrow\rightarrow)^2\diamond \ . \end{aligned}$$

If we want to encode also the edge labels in Figure 2.4, then we can exploit the fact that each node has exactly one outgoing edge: we label each position of ζ_2 by an edge label in addition (or instead) of a Dutch word. □

3 Training and parsing algorithms for probabilistic IRTG

When modeling phenomena of natural language, we are often faced with *ambiguity*. By *ambiguity*, we mean that some utterance of natural language (e.g., a sound, a particular morphological inflection of a word, a sentence, or an entire text) can be explained in multiple ways. In the case of homophones such as "site" and "sight", different words evoke the same sound when spoken. A word form such as "plays" could be a verb as in "she plays" or a plural noun as in "she acted in plays". The sentence "She saw the astronomer with the telescope." allows for different attachments of the prepositional phrase "with the telescope": either "the astronomer" has a telescope or the act of "seeing" is mediated by it. A sentence may be used with the same syntactic interpretation to mean the opposite things: e.g., "Well done." might be used ironically.

Currently, the field of natural language processing often addresses these ambiguities by viewing them as uncertainty and enhancing models with probabilistic components, but ideas of utilizing probabilistic models have been around at least since the work of Suppes (1972). Conceptionally, each of the different explanations of a particular utterance shall be assigned the probability that this explanation is correct. In order to obtain such a model one can, for instance, define a joint probability distribution over all explanations and utterances and then marginalize it for a particular utterance.

Although this might not be the closest reflection of how humans process language, practical models have been developed following this approach. In this section, we consider models based on IRTG extended by probabilities. Each derivation d of the IRTG has a probability, which is the product of the probabilities of the rules it contains. Each tree t in the tree language of the IRTG has the sum of the probabilities of the derivations that license it as probability. Finally, each domain object a has a probability, which is obtained by adding the probabilities of the trees that can be interpreted to a. Notably, there are two layers of ambiguity here: derivations are explanations for the trees, which again are explanations for domain objects. On the one hand, this causes us quite some trouble computationally when processing these grammars, because optimizing a sum of products resists dynamic programming. On the other hand, it allows us to define algorithms for obtaining subtle distributions over operator trees or domain objects.

In the following, we recall and adapt a wide range of theory that was originally developed for probabilistic context-free grammar. We present this material for probabilistic RTG and, in particular, probabilistic IRTG. Notably, probabilistic RTG are a special case of probabilistic context-free grammars, because every RTG is a CFG which generates a particular well-bracketed language.

Moreover, we present a generalization of the probabilistic context-free grammar with latent annotation framework, which was initially introduced by Matsuzaki, Miyao, and Tsujii (2005) and developed further by Petrov et al. (2006) and Petrov and Klein (2007), to IRTG. The material presented in this chapter is an extension of the work published at COLING (Gebhardt 2018).

3.1 Probabilistic interpreted regular tree grammars

We start by defining probabilistic RTG before turning to probabilistic IRTG. For all of Section 3.1 let $G = (N, \Sigma, S, R)$ be an RTG.

Definition 3.1.1. A *weight assignment for* G is a mapping $p\colon R \to [0,1]_{\mathbb{R}}$. Let $d \in (\mathrm{T}_R)_S$. The *weight of* d is

$$W_{(G,p)}(d) = \prod_{w \in \mathrm{pos}(d)} d(w) \ .$$

Let $t \in \mathrm{T}_\Sigma$. The *weight of* t is

$$W_{(G,p)}(t) = \sum_{d \in (\mathrm{T}_R)_S \colon [\![d]\!]_S^\Pi = t} W_{G,p}(d) \ .$$

If $\infty > \sum_{t \in \mathrm{T}_\Sigma} W_{(G,p)}(t)$, then p is called *convergent* and (G,p) is called *convergent weighted RTG*. If $1 = \sum_{\varrho \in R_B} p(\varrho)$ for each $B \in N$, then p is called *proper*. If $1 = \sum_{t \in \mathrm{T}_\Sigma} W_{(G,p)}(t)$, then we call p *consistent*. If p is proper and consistent, then we call p *probability assignment for* G and (G,p) *probabilistic RTG*. We denote the set of all probability assignments for G by $\mathcal{M}(G)$. □

Weighted RTGs that are convergent (often) induce probability distributions on T_Σ.

Definition 3.1.2. Let p be a convergent weight assignment for G where we have that $\sum_{t \in \mathrm{T}_\Sigma} W_{(G,p)}(t) > 0$. We define the following probability distributions where, for every $d \in (\mathrm{T}_R)_S$, we have

$$\mathrm{P}(d \mid G, p) = \frac{W_{(G,p)}(d)}{\sum_{d' \in (\mathrm{T}_R)_S} W_{(G,p)}(d')}$$

and, for every $t \in \mathrm{T}_\Sigma$, we have

$$\mathrm{P}(t \mid G, p) = \frac{W_{(G,p)}(t)}{\sum_{t' \in \mathrm{T}_\Sigma} W_{(G,p)}(t')} \ . \qquad \square$$

The following assumption reduces the notational effort for the remainder of this chapter.

Assumption 3.1.3. When we consider a convergent weighted RTG (G, p) in the following, we ignore the pathological case that $\sum_{t \in \mathrm{T}_\Sigma} W_{(G,p)}(t) = 0$. It occurs either because $L(G) = \emptyset$ or because, for each $d \in (\mathrm{T}_R)_S$, there is a position $w \in \mathrm{pos}(d)$ such that $p(d(w)) = 0$. □

Note that either case is not of particular interest for NLP applications. Usually, rules with zero probability or rules that do not occur in any derivation tree in $(\mathrm{T}_R)_S$ are removed (*trimmed*) from G. In a practical implementation these special cases are usually handled by separate routines.

Observation 3.1.4. If p is a probability assignment for G, then the equations in Definition 3.1.2 can be reduced such that, for every $d \in (\mathrm{T}_R)_S$, we have

$$\mathrm{P}(d \mid G, p) = W_{(G,p)}(d)$$

and, for every $t \in \mathrm{T}_\Sigma$, we have

$$\mathrm{P}(t \mid G, p) = W_{(G,p)}(t) \ . \qquad \square$$

Definition 3.1.5. Let (G, p) be a convergent weighted RTG. The *most probable tree of* (G, p) is

$$\arg\max_{t \in \mathrm{T}_\Sigma} \mathrm{P}(t \mid G, p) \ . \tag{3.1}$$

□

Example 3.1.6. Let $\Sigma = \{f_0, f_1, f_2\}$ be a ranked alphabet where $\mathrm{rk}(f_i) = i$ for each $i \in [2]_0$. Consider the RTG $G = (N, \Sigma, S, R)$ where $N = \{S, B\}$ and R contains rules ϱ with weights $p(\varrho)$ (behind the #) as follows:

$$\begin{array}{ll} S \to f_1(B) & \#1.0 \\ B \to f_2(B, B) & \#0.2 \\ B \to f_0() & \#0.8 \end{array}$$

Consider also the RTG $G' = (N', \Sigma, S, R')$ where $N' = \{S, B_1, B_2\}$ and R' contains rules ϱ with weights $p'(\varrho)$ as follows:

$$\begin{array}{llll} S \to f_1(B_1) & \#1.0 & B_1 \to f_0() & \#0.25 \\ B_1 \to f_2(B_1, B_2) & \#0.5 & B_2 \to f_0() & \#1.0 \\ B_1 \to f_2(B_2, B_1) & \#0.25 & & \end{array}$$

Observe that p and p' are proper. They are also consistent (without proof).

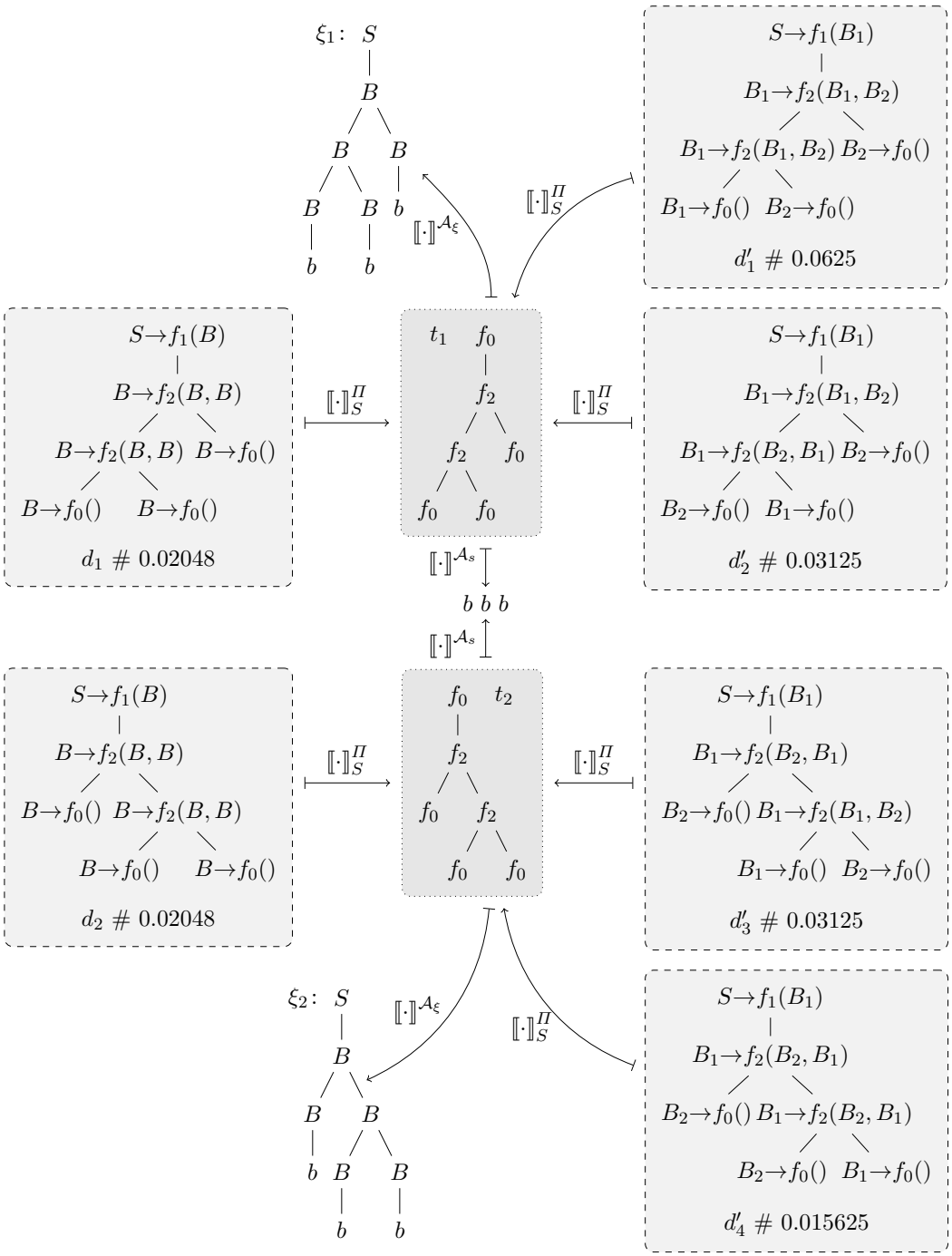


Figure 3.1: Derivation trees of RTGs G (left) and G' (right) with their probabilities; projection to operator trees (t_1, t_2); projection to a string (b b b) and parse trees (ξ_1, ξ_2).

Figure 3.1 shows derivation trees of G and G' and their probabilities. Their projection to trees over Σ is shown as well. Then

$$\begin{aligned}
\mathrm{P}(t_1 \mid G, p) &= \mathrm{P}(d_1 \mid G, p) &&= 0.02048\\
\mathrm{P}(t_2 \mid G, p) &= \mathrm{P}(d_2 \mid G, p) &&= 0.02048\\
\mathrm{P}(t_1 \mid G', p') &= \mathrm{P}(d'_1 \mid G', p') + \mathrm{P}(d'_2 \mid G', p') &&= 0.09375\\
\mathrm{P}(t_2 \mid G', p') &= \mathrm{P}(d'_3 \mid G', p') + \mathrm{P}(d'_4 \mid G', p') &&= 0.046875
\end{aligned}$$

We observe that (G, p) assigns the same probability to t_1 and t_2, i.e., it cannot distinguish between a left-branching and a right-branching derivation structure. In contrast, (G', p') employs more nonterminals and rules and assigns higher probability to t_1 than to t_2. □

Definition 3.1.7. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG and let $p \in \mathcal{M}(G)$. We call $(\mathbb{G}, p)$ *probabilistic IRTG*. We define the probability distribution on $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ given $(\mathbb{G}, p)$, where, for every $a \in \mathcal{A}$,

$$\mathrm{P}(a \mid \mathbb{G}, p) = \sum_{t \in \mathrm{parses}_{\mathbb{G}}(a)} \mathrm{P}(t \mid G, p) \ . \qquad \square$$

We apply marginalization to define a probability distribution over subspaces of $\mathcal{A}$.

Definition 3.1.8. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG and let $p \in \mathcal{M}(G)$. Let $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ and let $I \subseteq [k]$. Let $a' \in \bigtimes_{i \in I} \mathcal{A}_i$.

$$\mathrm{P}(a' \mid \mathbb{G}, p) = \sum_{\substack{d \in (\mathrm{T}_R)_S : \\ [\![[\![d]\!]_S^{\Pi}]\!]^{(\mathcal{A}_i \mid i \in I)} = a'}} \mathrm{P}(d \mid G, p) \ . \tag{3.2}$$

□

Similarly, we define conditional probability distributions:

Definition 3.1.9. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG and let $p \in \mathcal{M}(G)$. Let $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$, $I \subseteq [k]$, and let $J = [k] \setminus I$. We define, for each $a' \in \mathcal{A}_I$ and $a'' \in \mathcal{A}_J$, the conditional probability of a'' given a':

$$\mathrm{P}(a'' \mid \mathbb{G}, p, a') = \frac{\mathrm{P}(a'', a' \mid \mathbb{G}, p)}{\mathrm{P}(a' \mid \mathbb{G}, p)} = \frac{1}{\mathrm{P}(a' \mid \mathbb{G}, p)} \sum_{\substack{d \in (\mathrm{T}_R)_S : \\ [\![[\![d]\!]_S^{\Pi}]\!]^{\mathcal{A}_I} = a' \\ [\![[\![d]\!]_S^{\Pi}]\!]^{\mathcal{A}_J} = a''}} \mathrm{P}(d \mid G, p) \ . \tag{3.3}$$

The *parsing problem for* $(\mathbb{G}, p)$ *given* $a' \in \mathcal{A}_I$ is to compute

$$\hat{a} = \arg\max_{a'' \in \mathcal{A}_J} \mathrm{P}(a'' \mid a', \mathbb{G}, p) \ . \tag{3.4}$$

We call $\hat{a}$ the *most probable parse of* $(\mathbb{G}, p)$ *for* a'. □

Example 3.1.10 (Example 3.1.6 cont'd)**.** Consider the $\{\iota\}$-sorted alphabet $(\Sigma, \mathrm{sort}_\Sigma)$ and the $(\Sigma, \mathrm{sort}_\Sigma)$-algebras $\mathcal{A}_s$ and $\mathcal{A}_\xi$ where $(\mathcal{A}_s)_\iota = \{b\}^*$, $(\mathcal{A}_\xi)_\iota = \mathrm{U}_\Sigma$, and

$$\begin{aligned}
&\mathrm{sort}(f_0) = (\varepsilon, \iota) && f_0^{\mathcal{A}_s}() = b && f_0^{\mathcal{A}_\xi} = B(b)\\
&\mathrm{sort}(f_1) = (\iota, \iota) && f_0^{\mathcal{A}_s}(x_1) = x_1 && f_0^{\mathcal{A}_\xi} = S(x_1)\\
&\mathrm{sort}(f_2) = (\iota\iota, \iota) && f_0^{\mathcal{A}_s}(x_1, x_2) = x_1 \cdot x_2 && f_0^{\mathcal{A}_\xi} = B(x_1, x_2)\ .
\end{aligned}$$

Figure 3.1 shows the interpretation of two operator trees t_1 and t_2 in T_Σ to the same string $w \in \{b\}^*$ in the algebra $\mathcal{A}_s$. On the other hand, the operator trees are interpreted to different unranked *parse trees* ξ_1 and ξ_2 in the algebra $\mathcal{A}_\xi$.

We return to the big picture by considering the IRTGs $\mathbb{G} = (G, \mathcal{A}_s, \mathcal{A}_\xi)$ and $\mathbb{G}' = (G', \mathcal{A}_s, \mathcal{A}_\xi)$ with probability assignments p and p', respectively. We obtain the following probabilities for bbb, (bbb, ξ_1), and (bbb, ξ_2):

$$\begin{aligned}
\mathrm{P}(bbb \mid \mathbb{G}, p) &= \mathrm{P}(d_1 \mid G, p) + \mathrm{P}(d_2 \mid G, p) && = 0.04096\\
\mathrm{P}(bbb \mid \mathbb{G}', p') &= \sum_{i=1}^{4} \mathrm{P}(d_i' \mid G', p') && = 0.140626\\
\mathrm{P}((bbb, \xi_1) \mid \mathbb{G}, p) &= \mathrm{P}(d_1 \mid G, p) = 0.02048 && = \mathrm{P}(d_2 \mid G, p) = \mathrm{P}((bbb, \xi_2) \mid \mathbb{G}, p)\\
\mathrm{P}((bbb, \xi_1) \mid \mathbb{G}', p') &= \mathrm{P}(t_1 \mid G', p') = 0.09375 && \\
&> 0.046875 = \mathrm{P}(t_2 \mid G', p') = \mathrm{P}((bbb, \xi_2) \mid \mathbb{G}', p')\ .
\end{aligned}$$

Concerning conditional probability distributions induced by the probabilistic IRTG we have that

$$\begin{aligned}
\mathrm{P}(\xi_1 \mid bbb, \mathbb{G}, p) &= \frac{\mathrm{P}(d_1 \mid G, p)}{\mathrm{P}(d_1 \mid G, p) + \mathrm{P}(d_2 \mid G, p)} = 0.5 = \mathrm{P}(\xi_2 \mid bbb, \mathbb{G}, p)\\
\mathrm{P}(\xi_1 \mid bbb, \mathbb{G}', p') &= \frac{\mathrm{P}(t_1 \mid G', p')}{\mathrm{P}(bbb \mid \mathbb{G}', p')} = \frac{2}{3}\\
\mathrm{P}(\xi_2 \mid bbb, \mathbb{G}', p') &= \frac{\mathrm{P}(t_2 \mid G', p')}{\mathrm{P}(bbb \mid \mathbb{G}', p')} = \frac{1}{3}\ .
\end{aligned}$$

The most probable parse of $(\mathbb{G}, p)$ given bbb is ambiguous, as both (bbb, ξ_1) and (bbb, ξ_2) get assigned the same probability. In contrast, ξ_2 is the unique most probable parse of $(\mathbb{G}', p')$ given bbb. □

For technical and practical considerations, we also define the conditional probability of a derivation tree d given a domain object and a probabilistic IRTG.

Definition 3.1.11. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG, $p \in \mathcal{M}(G)$, $d \in (\mathrm{T}_R)_S$, and $a \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$. We define

$$\mathrm{P}(d \mid a, \mathbb{G}, p) = \begin{cases} \frac{\mathrm{P}(d|G,p)}{\mathrm{P}(a|\mathbb{G},p)} & \text{if } \left[\!\left[[\![d]\!]_S^{\Pi} \right]\!\right]^{\mathcal{A}_{[k]}} = a \\ 0 & \text{otherwise.} \end{cases}$$

□

A central question when modelling probability distributions by means of IRTG is to choose a good probability assignment. The remainder of this chapter is mainly devoted to how this challenging task can be accomplished. We start by recalling some basic definitions and properties.

Definition 3.1.12. Let $d \in (\mathrm{T}_R)_S$ and $B \in N$. The *number of occurrences of B in d*, denoted by $\mathrm{occ}_B(d)$, is

$$\mathrm{occ}_B(d) = |\{p \in \mathrm{pos}(d) \mid d(p) \in R_B\}| \ .$$

Let $\varrho \in R$. Then *number of occurrences of ϱ in d*, denoted by $\mathrm{occ}_\varrho(d)$, is

$$\mathrm{occ}_\varrho(d) = |\{p \in \mathrm{pos}(d) \mid d(p) = \varrho\}| \ . \qquad \square$$

If we are given an RTG G and a distribution q over derivation trees of G, then it is easy to select the probability assignment p for G that maximizes the likelihood of q under $\lambda d.\mathrm{P}(d \mid G, p)$ (or, equivalently minimizes KL-divergence between q and $\lambda d.\mathrm{P}(d \mid G, p)$).

Lemma 3.1.13 (Corazza and Satta 2006)**.** Let $G = (N, \Sigma, S, R)$ be an RTG. Let $q\colon (\mathrm{T}_R)_S \to [0,1]_{\mathbb{R}}$ be a probability distribution. Then

$$p^* = \operatorname*{arg\,max}_{p \in \mathcal{M}(G)} \mathrm{KL}\left(\lambda d.\mathrm{P}(d \mid G, p) \mid\mid q\right)$$

is such that, for each ϱ in R of the form $B \to \sigma(B_1, \ldots, B_n)$, we have

$$p^*(\varrho) = \frac{\mathbb{E}_q\left[\mathrm{occ}_\varrho\right]}{\mathbb{E}_q\left[\mathrm{occ}_B\right]} \ ,$$

if this quotient is defined. $\square$

In the above lemma, the quotient is undefined for each nonterminal symbol B that does not occur in the data, i.e., for each derivation d where $\mathrm{occ}_B(d) > 0$ we have $q(d) = 0$. The weights of rules in R_B are irrelevant when the likelihood and related quantities are computed. Hence, in the following we mostly ignore this degenerated case. However, should one need to handle this case, one can proceed as in the next definition. Given an arbitrary corpus over R, a probability distribution on R can be obtained by normalization:

Definition 3.1.14. Let $c\colon R \to \mathbb{R}_{\geq 0}$. We define the *normalization of c*, denoted by $\mathrm{norm}(c)$, to be the probability assignment where, for each rule ϱ of the form $B \to \sigma(B_1, \ldots, B_k)$, we have

$$(\mathrm{norm}(c))(\varrho) = \begin{cases} \frac{c(\varrho)}{\sum_{\varrho' \in R_B} c(\varrho')} & \text{if } \sum_{\varrho' \in R_B} c(\varrho') > 0 \\ \frac{1}{|R_B|} & \text{otherwise.} \end{cases}$$

$\square$

3.2 Expectation/maximization training for IRTG

The *expectation/maximization algorithm* (short EM algorithm) was introduced by Dempster, Laird, and Rubin (1977) as a way to find the maximum likelihood estimate from incomplete data under certain model restrictions. In the context of probabilistic RTG and probabilistic IRTG, this translates to finding the best probability assignment for a given grammar when just a corpus over trees and a corpus over the domain, respectively, is observed. To solve this problem, we would like to apply Lemma 3.1.13 but are given incomplete information instead of the required distribution over derivation trees.

The idea of Dempster, Laird, and Rubin is to estimate a distribution over the complete data based on the incomplete data and an initially guessed probability distribution that adheres to the model restrictions. For probabilistic (I)RTG, this implies the estimation of a distribution over derivation trees given an initial probability assignment.

More generally, in the realm of probabilistic grammars, the EM algorithm has been instantiated for, e.g., Hidden Markov Models (called: *Baum-Welch algorithm*; cf. Baum et al. 1970) and Probabilistic Context-free Grammars (called: inside-outside algorithm; cf. Baker 1979; Lari and Young 1990). In the following, we present a version of the inside-outside algorithm for IRTG.

3.2.1 Inside and outside weights

A concept closely connected to the variant of the EM algorithm for probabilistic grammars are inside and outside weights. These weights intuitively capture the probability mass that rests on a particular nonterminal symbol. The inside weight equals the sum of the probabilities of sub-derivations that start in a particular nonterminal symbol. The outside weights equal the sum of the probabilities of partial derivations that end in a particular nonterminal symbol. Both concepts date back to Baker (1979) and Lari and Young (1990) and are a generalization of the forward and backward weights for Hidden Markov Models (Baum et al. 1970). A treatment for probabilistic tree automata of the inside weight and its applications is provided by Maletti and Satta (2009). Nederhof and Satta (2008) consider so-called *partition functions* (another name for inside weights) and different ways to compute them for PCFG where the total probability mass is ≤ 1.

Definition 3.2.1. Let $G = (N, \Sigma, S, R)$ be an RTG and $p\colon R \to [0,1]_{\mathbb{R}}$ be a weight assignment for G. For $A, B \in N$, we define the set of *B-partial derivation trees rooted in A*, denoted by $(\mathrm{T}^{\mathrm{p}}_R(B))_A$, to be the set

$$\{d \in (\mathrm{T}_R(\{B\}))_A \mid 1 = |\mathrm{pos}_{\{B\}}(d)|\}$$

where $\mathrm{sort}(B) = B$. We extend, for each $c \in (\mathrm{T}^{\mathrm{p}}_R(B))_A$, the definition of $W_{(G,p)}$ such

that

$$W_{(G,p)}(c) = \prod_{w \in \mathrm{pos}_R(c)} p(c(w)) \ .$$

The *inside weight* $\beta_{(G,p)}(B)$ and the *outside weight* $\alpha_{(G,p)}(B)$ of a nonterminal $B \in N$ with respect to (G,p) are defined as

$$\beta_{(G,p)}(B) = \sum_{d \in (\mathrm{T}_R)_B} W_{(G,p)}(d)$$

and

$$\alpha_{(G,p)}(B) = \sum_{c \in (\mathrm{T}^{\mathrm{p}}_R(B))_S} W_{(G,p)}(c) \ \text{, respectively.}$$

For any rule ϱ of the form $B \to \sigma(B_1, \ldots, B_k)$, we let $\alpha_{(G,p)}(\varrho) = \alpha_{(G,p)}(B)$ and $\beta_{(G,p)}(\varrho) = \beta_{(G,p)}(B_1) \cdot \ldots \cdot \beta_{(G,p)}(B_k)$. □

By decomposing the derivations of weighted RTG where we utilize the commutativity of addition and the distributivity of multiplication over addition, we obtain a system of equations that characterizes inside and outside weights in an alternative way.

Observation 3.2.2. Let $G = (N, \Sigma, S, R)$ be an RTG and p be a weight assignment for G. It holds that

$$\beta_{(G,p)}(B) = \sum_{B \to \sigma(B_1,\ldots,B_k) \in R_B} p(B \to \sigma(B_1, \ldots, B_k)) \cdot \beta_{(G,p)}(B_1) \cdot \ldots \cdot \beta_{(G,p)}(B_k)$$

and

$$\alpha_{(G,p)}(B) = \delta^S_B + \sum_{\substack{C \to \sigma(B_1,\ldots,B_k) \text{ in } R \\ i \in [k] \colon B_i = B}} \alpha_{(G,p)}(C) \cdot p(C \to \sigma(B_1, \ldots, B_k)) \cdot \prod_{j \in [k] \colon j \neq i} \beta_{(G,p)}(B_j) \ ,$$

where δ^S_B is the Kronecker delta, i.e., $\delta^S_B = 1$ if $B = S$ and 0 otherwise. □

In the remainder of this section, we give several applications of inside and outside weights. First, we note that the inside weight of the initial nonterminal S equals the sum of the weights of all trees.

Observation 3.2.3. Let $G = (N, \Sigma, S, R)$ be an RTG and p be a weight assignment for G. It holds that

$$\sum_{t \in \mathrm{T}_\Sigma} W_{(G,p)}(t) = \sum_{t \in \mathrm{T}_\Sigma} \sum_{\substack{d \in (\mathrm{T}_R)_S \colon \\ [\![d]\!]^{\Pi}_S = t}} W_{(G,p)}(d) = \sum_{d \in (\mathrm{T}_R)_S} W_{(G,p)}(d) = \beta_{(G,p)}(S) \ . \quad \square$$

The next lemma links inside and outside weights of a convergent weighted RTG with the expected frequency of nonterminals and rules in the derivation trees it generates.

Lemma 3.2.4. Let $G = (N, S, \Sigma, R)$ be an RTG, p a convergent weight assignment for G, $B \in N$, and ϱ in R of the form $B \to \sigma(B_1, \ldots, B_k)$. Then

$$\mathbb{E}_{\lambda d.\mathrm{P}(d|G,p)}\left[\mathrm{occ}_B\right] = \alpha_{(G,p)}(B) \cdot \beta_{(G,p)}(B) / \beta_{(G,p)}(S) \tag{3.5}$$

and

$$\mathbb{E}_{\lambda d.\mathrm{P}(d|G,p)}\left[\mathrm{occ}_\varrho\right] = \alpha_{(G,p)}(B) \cdot p(\varrho) \cdot \beta_{(G,p)}(B_1) \cdot \ldots \cdot \beta_{(G,p)}(B_k) / \beta_{(G,p)}(S) \ . \tag{3.6}$$

□

Proof.

$$\begin{aligned}
&\mathbb{E}_{\lambda d.\mathrm{P}(d|G,p)}\left[\mathrm{occ}_B\right] \\
&= \sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_B(d) \\
&= \sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \sum_{\substack{c \in (\mathrm{T}^{\mathrm{p}}_R(B))_S \\ d' \in (\mathrm{T}_R)_B : \\ c[B/d'] = d}} 1 && (*) \\
&= \sum_{d \in (\mathrm{T}_R)_S} \sum_{\substack{c \in (\mathrm{T}^{\mathrm{p}}_R(B))_S \\ d' \in (\mathrm{T}_R)_B : \\ c[B/d'] = d}} \mathrm{P}(c[B/d'] \mid G, p) \\
&= \sum_{c \in (\mathrm{T}^{\mathrm{p}}_R(B))_S} \sum_{d \in (\mathrm{T}_R)_B} \mathrm{P}(c[B/d] \mid G, p) \\
&= \left(\sum_{c \in (\mathrm{T}^{\mathrm{p}}_R(B))_S} W_{(G,p)}(c) \right) \cdot \left(\sum_{d \in (\mathrm{T}_R)_B} W_{(G,p)}(d) \right) \cdot \left(\sum_{d' \in (\mathrm{T}_R)_S} W_{(G,p)}(d') \right)^{-1} \\
&= \alpha_{(G,p)}(B) \cdot \beta_{(G,p)}(B) / \beta_{(G,p)}(S)
\end{aligned}$$

Note that $(*)$ follows from the observation that a derivation d with $\mathrm{occ}_B(d)$ occurrences of B can be decomposed into a B-partial derivation tree rooted in S and a tree in $(\mathrm{T}_R)_B$ in $\mathrm{occ}_B(d)$ distinct ways. The proof for (3.6) is analogous. ■

The weight assignment of every convergent weighted RTG can be renormalized to obtain an equivalent probabilistic RTG (for similar results for PCFG cf. Abney, McAllester, and Pereira 1999; Chi 1999; Nederhof and Satta 2003). The construction utilizes inside weights.

Theorem 3.2.5 (Maletti and Satta 2009, Def. 3, proof of Thm. 4)**.** Let (G, p) be a convergent weighted RTG where $G = (N, \Sigma, S, R)$. There is a probability assignment p' for G where, for each $d \in (\mathrm{T}_R)_S$, it holds that $\mathrm{P}(d \mid G, p) = \mathrm{P}(d \mid G, p')$. □

Computation of inside and outside weights. Let (G, p) be a convergent weighted RTG with $G = (N, \Sigma, S, R)$ and let $\prec'$ be such that

$$\prec' = \{(B_i, B) \mid B \to \sigma(B_1, \ldots, B_k) \text{ in } R, i \in [k]\} \ .$$

If the transitive closure of $\prec'$ can be extended to a linear order $\prec$ on N, then $\beta_{(G,p)}(B)$ depends on $\beta_{(G,p)}(B')$ only if $B' \prec B$. In this case we can compute $\beta_{(G,p)}(B)$ in the order of $\prec$ and $\alpha_{(G,p)}(B)$ in the reverse order of $\prec$.

Otherwise, Observation 3.2.2 provides a system of polynomial equations with positive coefficients that can be solved analytically in certain cases (if the degree of the polynomials is reasonably small). In general, there are numerically options to solve it. One way that uses the fixed point theorem by Knaster (1928) and Tarski (1955) is discussed for probabilistic tree automata by Maletti and Satta (2009). Alternatively, gradient descent approaches using Newton's method or Broyden's method may be used as outlined by Nederhof and Satta (2008). Finally, using the fixed point theorem one can also show the following result:

Lemma 3.2.6. Let G be an RTG and p a weight assignment for G. If p is proper, then p is also convergent. □

3.2.2 Grammar morphisms

Although we can compute inside and outside weights of an arbitrary RTG G with convergent weight assignment p, we will benefit most from these concepts in the IRTG framework when looking at the RTG G_a that is obtained by intersecting $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ with $D(a)$ for some domain object $a \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$. Due to the nature of the intersection construction, we will assume that the nonterminals of G_a can be related to those in G in a way that is faithful to the rules. Secondly, we are also going to consider structurally similar grammars that differ in the granularity of their respective sets of nonterminals. We formalize these relationships by so-called *grammar morphisms*.

Definition 3.2.7. Let $G = (N, \Sigma, S, R)$ and $G' = (N', \Sigma, S', R')$ be RTGs. Let $\varphi\colon N' \to N$. We lift φ to $\bar{\varphi}\colon R[N', \Sigma] \to R[N, \Sigma]$ by setting

$$\bar{\varphi}(B' \to \sigma(B'_1, \ldots B'_n)) = \varphi(B') \to \sigma(\varphi(B'_1), \ldots, \varphi(B'_n)) \ .$$

If $\varphi^{-1}(S) = \{S'\}$ and $\bar{\varphi}(R') \subseteq R$, then φ is called *grammar morphism from G' to G*. We may write φ instead of $\bar{\varphi}$. □

We reconsider our running example to illustrate a grammar morphism.

Example 3.2.8 (Example 3.1.6 cont'd)**.** Consider the mapping from $\varphi\colon N' \to N$ where $\varphi(B_1) = B$, $\varphi(B_2) = B$, and $\varphi(S) = S$. Observe that $\varphi(R') = \varphi(R)$. Hence, φ is a grammar morphism from G' to G. □

We can also extend mappings φ from a set of nonterminals to some arbitrary alphabet M to grammar morphisms by constructing a grammar with nonterminals M appropriately.

Example 3.2.9. Let $G = (N, \Sigma, S, R)$ be an RTG, let M be an alphabet, and let $\varphi\colon N \to M$ such that $\varphi^{-1}(\varphi(S)) = \{S\}$. We construct a new RTG G' from G where we set $G' = (\varphi(N), \Sigma, \varphi(S), \varphi(R))$. Obviously, φ is a grammar morphism from G to G'. It is easy to see that $L(G) \subseteq L(G')$. In the following, we refer to G' by $\varphi(G)$. □

Throughout the thesis we assume a grammar morphism that gives rise to a one-to-one correspondence between the derivation trees of the chart G_a and the derivation trees of $\mathbb{G}$ for an object a of $\mathbb{G}$'s domain.

Assumption 3.2.10. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ with $G = (N, \Sigma, S, R)$ be an IRTG, let $a \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$, and $G_a = (N', \Sigma, S', R')$.

(a) We assume that there is a grammar morphism from G_a to G that we denote by φ_a^G. We extend φ_a^G to a tree homomorphism $\hat{\varphi_a^G}\colon (\mathrm{T}_{R'}) \to (\mathrm{T}_R)$ where we let $\hat{\varphi_a^G}(\varrho(d_1, \ldots, d_m)) = \varphi_a^G(\varrho)(\hat{\varphi_a^G}(d_1), \ldots, \hat{\varphi_a^G}(d_m))$ for each $\varrho \in (R'_{(B_1 \cdots B_m, B)})$.

(b) We assume that G_a and φ_a^G are such that $\hat{\varphi_a^G}$ is a bijection between $(\mathrm{T}_{R'})_{S'}$ and $\left\{ d \in (\mathrm{T}_R)_S \;\middle|\; \left[\!\left[[\![d]\!]_S^{\Pi} \right]\!\right]^{\mathcal{A}_{[k]}} = a \right\}$.

In the following, we may write φ_a^G instead of $\hat{\varphi_a^G}$. □

Observation 3.2.11. Let $G = (N, \Sigma, S, R)$ and $G' = (N', \Sigma, S', R')$ be RTGs, let φ be a grammar morphism from G' to G, and let p be a weight assignment for G. Then $p \circ \bar{\varphi}\colon R' \to [0, 1]_{\mathbb{R}}$ is a weight assignment for G'. □

Lemma 3.2.12. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG, let $a \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$, and let p be a probability assignment for G. Then

$$\mathrm{P}(a \mid \mathbb{G}, p) = \sum_{t \in L(G_a)} W_{(G_a,\, p \circ \varphi_a^G)}(t) \ . \qquad \square$$

Proof. Let $G_a = (N', \Sigma, S', R')$. It holds that

$$\begin{aligned} \mathrm{P}(a \mid \mathbb{G}, p) &= \sum_{t \in \mathrm{parses}_{\mathbb{G}}(a)} W_{(G,p)}(t) \\ &= \sum_{t \in \mathrm{parses}_{\mathbb{G}}(a)} \sum_{\substack{d \in (\mathrm{T}_R)_S\colon \\ [\![d]\!]_S^{\Pi} = t}} \prod_{w \in \mathrm{pos}(d)} p(d(w)) \end{aligned}$$

$$= \sum_{t \in L(G_a)} \sum_{\substack{d' \in (\mathrm{T}_{R'})_{S'}: \\ [\![d']\!]^{\Pi}_{S'} = t}} \prod_{w \in \mathrm{pos}(d')} p((\varphi_a^G(d'))(w)) \qquad \text{(Assumption 3.2.10)}$$

$$= \sum_{t \in L(G_a)} \sum_{\substack{d' \in (\mathrm{T}_{R'})_{S'}: \\ [\![d']\!]^{\Pi}_{S'} = t}} \prod_{w \in \mathrm{pos}(d')} (p \circ \varphi_a^G)(d'(w))$$

$$= \sum_{t \in L(G_a)} W_{(G_a,\, p \circ \varphi_a^G)}(t) \qquad \blacksquare$$

From the previous lemma and Observation 3.2.3 the following corollary follows. It provides a way of computing the probability of any domain object given a probabilistic IRTG.

Corollary 3.2.13. Let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG and p be a probability assignment for G. Then, for each $a \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ with $G_a = (N', \Sigma, S', R')$, we have

$$\mathrm{P}(a \mid \mathbb{G}, p) = \beta_{(G_a, p \circ \varphi_a^G)}(S')$$

and, for each d' in $(\mathrm{T}_{R'})_{S'}$, that

$$\mathrm{P}(\varphi_a^G(d') \mid a, \mathbb{G}, p) = \mathrm{P}(d' \mid G_a, p \circ \varphi_a^G) \ . \qquad \square$$

Proof. Note that $(G, \varphi_a^G \circ p)$ is convergent by Lemma 3.2.12. Then the first part is a straightforward consequence of Lemma 3.2.12 and Observation 3.2.3. For the second part, we have that

$$\begin{aligned}
\mathrm{P}(\varphi_a^G(d') \mid a, \mathbb{G}, p) &= \mathrm{P}(\varphi_a^G(d') \mid G, p) \cdot \mathrm{P}(a \mid \mathbb{G}, p)^{-1} && \text{(Definition 3.1.11)} \\
&= \prod_{w \in \mathrm{pos}(\varphi_a^G(d'))} p\left((\varphi_a^G(d'))(w)\right) \cdot \mathrm{P}(a \mid \mathbb{G}, p)^{-1} \\
& && \text{(Observation 3.1.4)} \\
&= \prod_{w \in \mathrm{pos}(d')} (p \circ \varphi_a^G)(d'(w)) \cdot \mathrm{P}(a \mid \mathbb{G}, p)^{-1} \\
&= W_{(G_a, p \circ \varphi_a^G)}(d') \cdot \left(\sum_{t \in L(G_a)} W_{(G_a, p \circ \varphi_a^G)}(t) \right)^{-1} && \text{(Lemma 3.2.12)} \\
&= \mathrm{P}(d' \mid G_a, p \circ \varphi_a^G) && \text{(Definition 3.1.2)}
\end{aligned}$$

$\blacksquare$

3.2.3 The EM-Algorithm

The expectation/maximization (EM) algorithm (Dempster, Laird, and Rubin 1977) in the inside-outside variant (Baker 1979; Lari and Young 1990) carries over to IRTGs. Let $\mathbb{G} = (G, \mathcal{A}_1, \dots, \mathcal{A}_k)$ be an IRTG with $G = (N, \Sigma, S, R)$, let $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$, and let p_i be a probability assignment for G. Let $c_{\mathcal{A}} \colon \mathcal{A} \to [0,1]_{\mathbb{R}}$ be a probability distribution over the domain. Moreover, for each $a \in \mathcal{A}$, let $G_a = (N_{G_a}, \Sigma, S_{G_a}, R_{G_a})$. The EM-algorithm iterates two steps, the expectation and the maximization step:

Expectation step. We compute a corpus $c_i \colon R \to \mathbb{R}_{\geq 0}$ over rules such that, for each $\varrho \in R$, we have

$$c_i(\varrho) = \mathbb{E}_{c_{\mathcal{A}}}\left[\lambda a.\mathbb{E}_{\lambda d.\mathrm{P}(d|a,\mathbb{G},p)}\left[\mathrm{occ}_\varrho\right]\right] \quad . \tag{3.7}$$

Intuitively, we define a corpus c_i over the rules of G such that each rule ϱ is assigned the frequency with which ϱ is expected to be used to generate the training data $c_{\mathcal{A}}$. By employing the assumptions concerning the charts G_a and inside and outside weights, we can effectively compute c_i by accumulating expected rule frequencies over all charts.

Lemma 3.2.14. For each $\varrho \in R$ it holds that

$$c_i(\varrho) = \textstyle\sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \sum_{\varrho' \in \varphi_a^{G\,-1}(\varrho)} \dfrac{\alpha_{(G_a, p\circ\varphi_a^G)}(\varrho') \cdot (p \circ \varphi_a^G)(\varrho') \cdot \beta_{(G_a, p\circ\varphi_a^G)}(\varrho')}{\beta_{(G_a, p\circ\varphi_a^G)}(S_{G_a})} \quad . \quad \square$$

Proof.

$$\begin{aligned}
c_i(\varrho) &= \mathbb{E}_{c_{\mathcal{A}}}\left[\lambda a.\mathbb{E}_{\lambda d.\mathrm{P}(d|a,\mathbb{G},p)}\left[\mathrm{occ}_\varrho\right]\right] \\
&= \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid a, \mathbb{G}, p) \cdot \mathrm{occ}_\varrho(d) \\
&= \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \sum_{d' \in (\mathrm{T}_{R'})_{S'}} \mathrm{P}(d' \mid G_a, p \circ \varphi_a^G) \cdot \mathrm{occ}_\varrho(\varphi_a^G(d')) \qquad \text{(Corollary 3.2.13)} \\
&= \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \sum_{\varrho' \in \varphi_a^{G\,-1}(\varrho)} \frac{\alpha_{(G_a, p\circ\varphi_a^G)}(\varrho') \cdot (p \circ \varphi_a^G)(\varrho') \cdot \beta_{(G_a, p\circ\varphi_a^G)}(\varrho')}{\beta_{(G_a, p\circ\varphi_a^G)}(S_a^G)} \\
&\qquad\qquad \text{(Lemma 3.2.4)}
\end{aligned}$$

■

Maximization step. The updated probability assignment p_{i+1} is chosen such that the likelihood of c_i under p_{i+1} is maximized, i.e.,

$$p_{i+1} = \operatorname*{arg\,max}_{p' \colon \text{proper weight assignment for } G} \left(\sum_{\varrho \in R} c_i(\varrho) \cdot \log(p'(\varrho)) \right) \quad .$$

Lemma 3.2.15 (Special case of Thm. 1 in Prescher 2004)**.** Let $c\colon R \to \mathbb{R}_{\geq 0}$. Then

$$\mathrm{norm}(c) = \operatorname*{arg\,max}_{p'\colon \text{proper weight assignment for } G} \left(\sum_{\varrho \in R} c(\varrho) \cdot \log(p'(\varrho)) \right) \quad . \qquad \square$$

Next we present the main theorem concerning the EM algorithm. It implies that the likelihood of the training data $c_{\mathcal{A}}$ under the probability assignment p_{i+1} resulting from an iteration of the EM algorithm is at least as high as under the probability assignment p_i. In addition it can be shown that the process converges to a local optimum or saddle point of the likelihood function (C. F. J. Wu 1983).

Theorem 3.2.16.

$$\sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \log(\mathrm{P}(a \mid \mathbb{G}, p_{i+1})) \geq \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \log(\mathrm{P}(a \mid \mathbb{G}, p_i)) \qquad \square$$

Before we prove Theorem 3.2.16, consider the following observation.

Observation 3.2.17. Let p be a proper and consistent probability assignment for G. Then for each family of probability distributions $q = (q_a\colon (\mathrm{T}_R)_S \to [0,1]_{\mathbb{R}} \mid a \in \mathcal{A})$ we have

$$\begin{aligned}
&\sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \log(\mathrm{P}(a \mid \mathbb{G}, p)) \\
&= \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \log \left(\sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(a, d \mid \mathbb{G}, p) \right) \\
&= \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \log \left(\sum_{d \in (\mathrm{T}_R)_S} q_a(d) \cdot \frac{\mathrm{P}(a, d \mid \mathbb{G}, p)}{q_a(d)} \right) \\
&\geq \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \sum_{d \in (\mathrm{T}_R)_S} q_a(d) \cdot \log \left(\frac{\mathrm{P}(a, d \mid \mathbb{G}, p)}{q_a(d)} \right) \qquad \text{(Jensen's inequality)}
\end{aligned}$$

We note that the last inequality is an **equality** if, for each a in $\mathcal{A}$, there is $x_a \in \mathbb{R}$ such that, for each $d \in (T_R)_S$, we have that $x_a = \frac{\mathrm{P}(a,d|\mathbb{G},p)}{q_a(d)}$. $\square$

Proof of Theorem 3.2.16. Let $q^* = (q_a^*\colon (\mathrm{T}_R)_S \to [0,1]_{\mathbb{R}} \mid a \in \mathcal{A})$ such that, for each $a \in \mathcal{A}$, we have $q_a^*(d) = \mathrm{P}(a, d \mid \mathbb{G}, p_i) / \sum_{d' \in (\mathrm{T}_R)_S} \mathrm{P}(a, d' \mid \mathbb{G}, p_i)$.

For each $a \in \mathcal{A}$, we choose a constant x_a such that $x_a = \sum_{d' \in (\mathrm{T}_R)_S} \mathrm{P}(a, d' \mid \mathbb{G}, p_i)$. Note that for each $d \in (\mathrm{T}_R)_S$ we have that

$$x_a = \sum_{d' \in (\mathrm{T}_R)_S} \mathrm{P}(a, d' \mid \mathbb{G}, p_i) = \frac{\mathrm{P}(a, d \mid \mathbb{G}, p_i)}{\left(\frac{\mathrm{P}(a,d|\mathbb{G},p_i)}{\sum_{d' \in (\mathrm{T}_R)_S} \mathrm{P}(a,d'|\mathbb{G},p_i)} \right)} = \frac{\mathrm{P}(a, d \mid \mathbb{G}, p_i)}{q_a^*(d)} \quad .$$

Hence, from Observation 3.2.17 (in the equality version), it follows that

$$\sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\log(\mathrm{P}(a\mid\mathbb{G},p_i)) = \sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\sum_{d\in(\mathrm{T}_R)_S} q_a^*(d)\cdot\log\left(\frac{\mathrm{P}(a,d\mid\mathbb{G},p_i)}{q_a^*(d)}\right) \quad . \tag{3.8}$$

Now let

$$p^* = \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \quad \sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\sum_{d\in(\mathrm{T}_R)_S} q_a^*(d)\cdot\log\left(\frac{\mathrm{P}(a,d\mid\mathbb{G},p')}{q_a^*(d)}\right) \quad .$$

We obtain that

$$\begin{aligned}
&\sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\log(\mathrm{P}(a\mid\mathbb{G},p^*)) \\
&\geq \sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\sum_{d\in(\mathrm{T}_R)_S} q_a^*(d)\cdot\log\left(\frac{\mathrm{P}(a,d\mid\mathbb{G},p^*)}{q_a^*(d)}\right) && \text{(Observation 3.2.17)}\\
&\geq \sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\sum_{d\in(\mathrm{T}_R)_S} q_a^*(d)\cdot\log\left(\frac{\mathrm{P}(a,d\mid\mathbb{G},p_i)}{q_a^*(d)}\right) && \text{(arg max)}\\
&= \sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\cdot\log(\mathrm{P}(a\mid\mathbb{G},p_i)) \quad . && \text{(cf. (3.8))}
\end{aligned}$$

It remains to show that $p_{i+1} = p^*$. We observe that $q_a^*(d) = \mathrm{P}(d\mid a,\mathbb{G},p)$. Then

$$\begin{aligned}
p_{i+1} &= \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \left(\sum_{\varrho\in R} c_i(\varrho)\cdot\log(p'(\varrho))\right)\\
&= \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \left(\sum_{\varrho\in R} \mathbb{E}_{c_{\mathcal{A}}}\left[\lambda a.\mathbb{E}_{\lambda d.\mathrm{P}(d\mid a,\mathbb{G},p)}\left[\mathrm{occ}_\varrho\right]\right]\cdot\log(p'(\varrho))\right)\\
&= \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \left(\sum_{\varrho\in R}\sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\sum_{d\in(\mathrm{T}_R)_S}\mathrm{P}(d\mid a,\mathbb{G},p)\cdot\mathrm{occ}_\varrho(d)\cdot\log(p'(\varrho))\right)\\
&= \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \left(\sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\sum_{d\in(\mathrm{T}_R)_S}\mathrm{P}(d\mid a,\mathbb{G},p)\cdot\sum_{\varrho\in R}\mathrm{occ}_\varrho(d)\cdot\log(p'(\varrho))\right)\\
&= \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \left(\sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\sum_{d\in(\mathrm{T}_R)_S}\mathrm{P}(d\mid a,\mathbb{G},p)\cdot\sum_{w\in\mathrm{pos}(d)}\log(p'(d(w)))\right)\\
&= \operatorname*{arg\,max}_{\substack{p'\colon \text{proper weight}\\ \text{assignment for } G}} \left(\sum_{a\in\mathcal{A}} c_{\mathcal{A}}(a)\sum_{d\in(\mathrm{T}_R)_S}\mathrm{P}(d\mid a,\mathbb{G},p)\cdot\log\left(\prod_{w\in\mathrm{pos}(d)} p'(d(w))\right)\right)
\end{aligned}$$

Algorithm 3.2.1 EM-Training

Input: IRTG $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$, probability assignment p_0 for G
corpus $c_{\mathcal{A}} \colon \mathcal{A} \to \mathbb{R}_{\geq 0}$
Output: probability assignment p' for G

1: $i \leftarrow 0$
2: **while** $i <$ max iterations **do**
3: $\quad c_i(\varrho) = \mathbb{E}_{c_{\mathcal{A}}}\left[\lambda a.\mathbb{E}_{\lambda d.\mathrm{P}(d|a,\mathbb{G},p_i)}\left[\mathrm{occ}_{\varrho}\right]\right]$ for each $\varrho \in R$
4: $\quad p_{i+1} = \underset{p' \colon \text{proper weight assignment for } G}{\arg\max} \left(\sum_{\varrho \in R} c_i(\varrho) \cdot \log(p'(\varrho))\right)$
5: $\quad i \leftarrow i + 1$
6: **output** p_i

$$
\begin{aligned}
&= \underset{\substack{p' \colon \text{proper weight} \\ \text{assignment for } G}}{\arg\max} \left(\sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \sum_{d \in (\mathrm{T}_R)_S} q^*_a(d) \cdot \log(\mathrm{P}(a, d \mid \mathbb{G}, p)) \right) \\
&= \underset{\substack{p' \colon \text{proper weight} \\ \text{assignment for } G}}{\arg\max} \left(\sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \sum_{d \in (\mathrm{T}_R)_S} q^*_a(d) \cdot \log\left(\frac{\mathrm{P}(a, d \mid \mathbb{G}, p)}{q^*_a(d)} \right) \right) \\
&= p^*
\end{aligned}
$$

■

In practice, the expectation and the maximization step are iterated until the difference of the likelihood of $c_{\mathcal{A}}$ under p_i and p_{i+1} falls below a certain threshold, until the likelihood of a validation corpus drops (for multiple epochs), or until a maximum number of iterations is reached. In Algorithm 3.2.1 we use the latter criterion. Our implementation additionally uses a validation corpus $c'_{\mathcal{A}}$ to stop EM training early if the likelihood of $c'_{\mathcal{A}}$ under p_{i+1} decreases.

3.2.4 Maximum a posteriori estimation with Dirichlet priors

Up to now we have described a training procedure for probabilistic IRTG that tries to select a probability assignment for the grammar that maximizes the likelihood of a training corpus given that probabilistic grammar. This optimization problem may also be described as follows:

$$p^* = \arg\max_{p \in \mathcal{M}(G)} \mathrm{P}(c_{\mathcal{A}} \mid G, p) \ .$$

However, one could also consider the following optimization problem:

$$p^* = \arg\max_{p \in \mathcal{M}(G)} \mathrm{P}(p \mid G, c_{\mathcal{A}})$$

$$= \arg\max_{p \in \mathcal{M}(G)} \frac{\mathrm{P}(c_{\mathcal{A}} \mid G, p) \cdot \mathrm{P}(p \mid G)}{\mathrm{P}(c_{\mathcal{A}} \mid G)} \qquad \text{(Bayes' rule)}$$
$$= \arg\max_{p \in \mathcal{M}(G)} \mathrm{P}(c_{\mathcal{A}} \mid G, p) \cdot \mathrm{P}(p \mid G) \ ,$$

which is called *maximum a posteriori* (MAP) estimation. We find the likelihood term $\mathrm{P}(c_{\mathcal{A}} \mid G, p)$ again to be part of the objective function, however, there is an additional term $\mathrm{P}(p \mid G)$ called *prior*. If we assume that each weight assignment for G is equally probable, then the maximum likelihood and the MAP objective coincide. In practice we may want to favor certain weight assignments or suppress others that we consider degenerate. For instance, if a certain rule of our grammar is never triggered by the training data, then the maximum likelihood objective selects a probability assignment that assigns weight 0 to it. Suppose that we added this rule to the grammar on purpose due to *prior knowledge* that we have about the data we intent to process. In this case it seems reasonable to prohibit probability assignments that set any rule weight to 0. In particular, we consider so-called *Dirichlet priors* following the approach of Johnson, Griffiths, and Goldwater (2007) for PCFG.

Probability assignments over continuous random variables. Next we want to define a prior on the set of probability assignments for G, which is an uncountable set and cannot be handled with the techniques outlined in Section 2.2. Hence, we briefly introduce a few notions to define probability distributions on subspaces of $\mathbb{R}^n$. This is sufficient because a probability assignment for $G = (N, \Sigma, S, R)$ can be seen as a family of real valued vectors $((u_1, \ldots, u_{|R_A|}) \mid A \in N)$ where each vector contains the rule probabilities of the rules with A on the left-hand side in an arbitrary but fixed order. Formally, for each $A \in N$, we have that $(u_1, \ldots, u_{|R_A|})$ is chosen from the $(|R_A| - 1)$-probability simplex $C_{|R_A|-1}$, where, for each $k \in \mathbb{N}$, we define $C_k = \{(u_0, \ldots, u_k) \in \mathbb{R}^{k+1}_{\geq 0} \mid \sum_{i=0}^{k} u_i = 1\}$. The k-probability simplex is a compact subspace of $\mathbb{R}^{k+1}$ over which we can integrate using a surface integral.

A *probability density over* C_k is a continuous function $f \colon C_k \to \mathbb{R}_{\geq 0}$ such that

$$\int_{C_k} f(x) \mathrm{dS}(x) = 1 \ .$$

Hence, we may define, for each subset X of C_k the probability $\mathrm{P}_f(X) = \int_X f(x) \mathrm{dS}(x)$. If we are interested in the most probable vector in C_k, then we are faced with the problem that $\mathrm{P}_f(x) = 0$ for each $x \in C_k$ because each point has the measure zero. However, if we consider for each point x a fixed-size ball $B_\varepsilon(x)$ with radius ε centered at x, then we obtain that

$$\lim_{\varepsilon \to 0} \arg\max_{x \in C_k} \mathrm{P}_f(B_\varepsilon(x)) = \arg\max_{x \in C_k} f(x) \ . \tag{3.9}$$

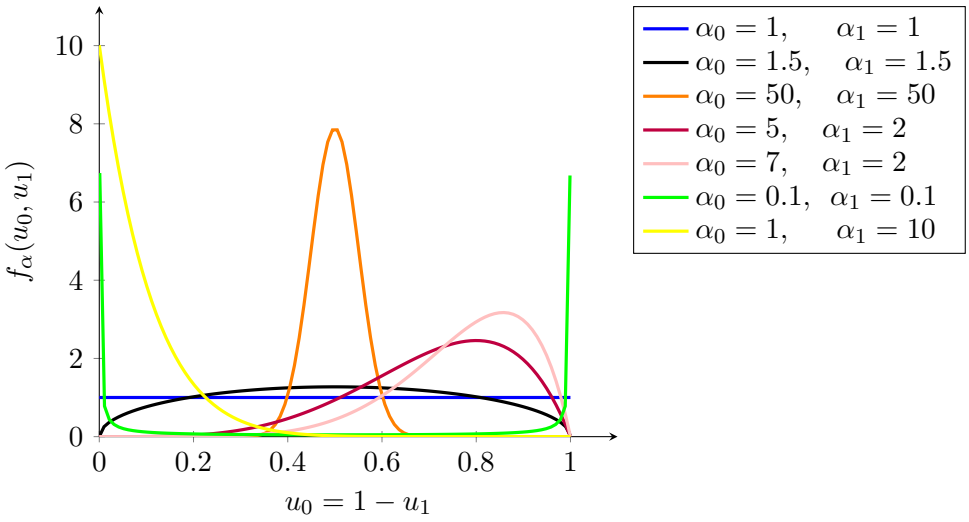


Figure 3.2: The probability density function for C_1 under different Dirichlet priors.

Dirichlet priors. As prior distributions we consider so-called *Dirichlet distributions* on the probability simplex (cf. Bishop 2006, Section 2.2.1). Let $\alpha = (\alpha_i \in \mathbb{R}_{>0} \mid i \in [k]_0)$ be a $[k]_0$-indexed family. We define the *Dirichlet probability density function on C_k parametrized by α*, denoted by f_α, such that

$$f_\alpha(u_0, \ldots, u_k) = Z_\alpha \prod_{i=0}^{k} u_i^{\alpha_i - 1}$$

where Z_α is a constant chosen such that $1 = \int_{C_k} f_\alpha(x) \mathrm{dS}(x)$. In particular,

$$Z_\alpha = \frac{\Gamma\left(\sum_{i=0}^{k} \alpha_i\right)}{\prod_{i=0}^{k} \Gamma(\alpha_i)}$$

where $\Gamma \colon \mathbb{R} \to \mathbb{R}$ is the *gamma*-function defined such that $\Gamma(x) = \int_0^\infty u^{x-1} e^{-u} \mathrm{d}u$. How different choices of α influence the probability density f_α is depicted in Figure 3.2. Notably, if $\alpha_i = 1$ for each $i \in [k]_0$, then each choice of u is equally likely. If $\alpha_i \gg a_{i'}$, then distributions that assign more probability mass to i than to i' are favored. If $\alpha_i \gg 1$ for each $i \in [k_0]$, then distributions that share the probability mass equally between all events are favored. On the contrary, if $\alpha_i \ll 1$ for each $i \in [k_0]$, then a vector u that concentrates all probability mass to one event is preferred.

The MAP estimation problem for selecting the best vector u given a corpus c over $[k]_0$ and a prior parameter α is as follows:

$$u^* = \arg\max_{u \in C_k} \mathrm{P}(u \mid c, \alpha) = \arg\max_{u \in C_k} \mathrm{P}(c \mid u) \cdot f_\alpha(u) \qquad \text{(Cf. (3.9))}$$

$$= \arg\max_{u \in C_k} \left(\prod_{i=0}^{k} u_i^{c(i)} \right) \cdot \left(Z_\alpha \cdot \prod_{i=0}^{k} u_i^{\alpha_i - 1} \right)$$
$$= \arg\max_{u \in C_k} Z_\alpha \prod_{i=0}^{k} u_i^{c(i)+\alpha_i-1}$$
$$= \arg\max_{u \in C_k} \prod_{i=0}^{k} u_i^{c(i)+\alpha_i-1}$$

Thus, a Dirichlet prior can be seen as a shift in the frequency with which an element occurs in the corpus. This makes it easy to incorporate in the EM training framework.

Assume that we have a certain distribution over rules, e.g., the one obtained in Equation (3.7). If we choose a Dirichlet prior parameter $\alpha_A \colon R_A \to \mathbb{R}$ for each set of rules with left-hand side nonterminal A, then Equation (3.7) needs to be modified as follows and we obtain a MAP training algorithm:

$$c_i(\varrho) = \mathbb{E}_{c_{\mathcal{A}}} \left[\lambda a. \mathbb{E}_{\lambda d. \mathrm{P}(d|a,\mathbb{G},p)} \left[\mathrm{occ}_\varrho \right] \right] + \alpha_A(\varrho) - 1 \ . \tag{3.10}$$

Note that the closed form solution for the maximization step as obtained by means of Lemma 3.2.15 requires that $c_i(\varrho) \geq 0$ for each $\varrho \in R$. A sufficient condition for this to hold in the MAP scenario is $\alpha_A(\varrho) \geq 1$ for each $\varrho \in R_A$. Hence, in the experiment we use a MAP-version of the EM algorithm with $\alpha_A(\varrho)$ set to different values above 1.0 ($\{1.1, 1.5, 2.0, 6.0, 11.0\}$) for each rule ϱ. In consequence, weight assignments that assign zero probability to some rule are ruled out.

3.3 Split/merge algorithm for IRTG

When probabilistic grammars, e.g., probabilistic IRTGs, shall be applied to solve NLP problems, we are faced with the problem of constructing the grammar and assigning weights to its rules. Apart from designing the rules by hand, the availability of large databases of utterances of natural language with linguistic annotations allows the development of algorithms that extract grammars automatically. These algorithms work by recursively decomposing a given entry in the database into parts. Each decomposition corresponds to a rule application. Each nonterminal shall capture properties of a particular part. One central concern in this process is choosing the relevant properties to encode into the nonterminals subject to performance considerations. Encoding too much information restricts the combinatorial potential of the rules while encoding too little information leads to *overgeneration*, i.e., rules may be connected to a derivation that is not plausible from a linguistic point of view.

To tackle this problem in an automatic fashion Matsuzaki, Miyao, and Tsujii (2005) suggested *probabilistic context-free grammar with latent annotation* (PCFG-LA). Starting from an overgenerating coarse PCFG G, they add to each nonterminal B a

latent annotation i from the set $[n]$ (where $n \in \mathbb{N}$) to obtain the refined nonterminal B_i. For each rule $A \to BC$ of G, there are now different refined versions $A_i \to B_j C_k$ for each possible choice of $i, j, k \in [n]$ each getting assigned a probability. The probability of a parse tree t, which still contains unrefined nonterminals, is defined as the sum over the probabilities over all refined versions of this parse tree. This way, a too restrictive grammar is avoided while still assigning a high probability to linguistically plausible rule combinations.

One disadvantage of Matsuzaki, Miyao, and Tsujii's (2005) model is that each nonterminal has the same number of latent annotations. This leads to large grammars because for each coarse rule that contains m nonterminals, there will be n^m refined rules. It is likely that not every nonterminal requires the same number of latent annotations. To this end, Petrov et al. (2006) extend the PCFG-LA model by an iterative refinement procedure that we call *split/merge algorithm* in the following. The algorithm takes an unrefined PCFG as input and applies multiple cycles (typically 6) composed of the following steps:

1. Each nonterminal is split into two new ones.
2. The probability assignment of the resulting grammar is refined by EM training.
3. 50% of the splits that are of less utility are undone.
4. The probability assignment of the resulting grammar is refined by EM training.
5. The probabilities of the rules are smoothed and refined by EM training once more.

Although the size of the resulting grammar is now exponential in the number of cycles, we can grant some nonterminals a large number of latent annotations that would be infeasible if granted to all nonterminals. Petrov et al. (2006) also note that this iterative training approach leads to better probability assignments.

The observation which is the foundation for this section is that PCFG-LA can be simulated by PRTG: In an RTG G, the nonterminal symbols do not occur in the generated tree language $L(G)$. The probability of a tree t in $L(G)$ corresponds to the sum of the probabilities of all derivation trees for t. Hence, this definition subsumes the probability of parse trees in the PCFG-LA model. Such a presentation (based on weighted tree automata) and a theoretic analysis of Petrov et al.'s 2006 method is due to Dietze (2018).

Our aim for this section is to extend the split/merge algorithm even further to probabilistic IRTG. Our refinement procedure will only alter the probabilistic RTG underlying the IRTG but does not change the algebras. In the following, we present a formalization of the steps of this generalized split/merge algorithm, where we follow the structure of Petrov et al. (2006) outlined above.

3.3.1 Splitting

The first step in each split/merge cycle is to split the nonterminals of a given *coarse* grammar. We define this step by means of a special grammar morphism μ that we call splitter. Note that, somewhat counterintuitively, the codomain of μ is the given grammar G. Hence, the split grammar is obtained by applying μ^{-1} to G.

Definition 3.3.1. Let $G = (N, \Sigma, S, R)$ be an RTG. A *splitter for* G is a surjective mapping $\mu\colon N' \to N$ where N' is a finite set (fine nonterminals) and $\mu^{-1}(S)$ is a singleton set, i.e., there is $S' \in N'$ such that $\mu^{-1}(S) = \{S'\}$. The *split of* G *with respect to* μ is the RTG $\mu^{-1}(G) = (N', \Sigma, S', R')$ where $R' = \{\varrho' \in \mathrm{R}[N', \Sigma] \mid \mu(\varrho') \in R\}$.

Let p be a probability assignment for G. We define a proper weight assignment p' for $\mu^{-1}(G)$ by setting, for each rule ϱ of the form $A \to \sigma(A_1, \ldots, A_k)$ in R',

$$p'(\varrho) = \frac{p(\mu(\varrho))}{\sum_{\varrho' \in R'_A} p(\mu(\varrho'))} \quad . \qquad \square$$

Observation 3.3.2. Let (G, p) be a probabilistic RTG with $G = (N, \Sigma, S, R)$, μ be a splitter for G, and p' be as in Definition 3.3.1. We have that μ is a grammar morphism from $G' = \mu^{-1}(G)$ to G and $p' = \mathrm{norm}(p \circ \mu)$. As a consequence of Lemma 3.2.6 we have that p' is also convergent. Thus, if p' is not consistent, we may obtain p'' such that p'' is a probability assignment for G' and the probability distributions of (G', p') and (G', p'') on $\mathrm{T}_{R'}$ and T_Σ are equivalent. In the following, w.l.o.g. we assume that p' is consistent.[1] $\square$

Next, we define a specific splitter that is based on Petrov et al. (2006). Petrov et al. (2006) split each nonterminal into two new nonterminals. As required by the definition of RTG, we will not split the initial nonterminal.[2]

Definition 3.3.3. Let $G = (N, \Sigma, S, R)$ be an RTG. The *2-splitter of* G is the grammar morphism $\mu_{\mathrm{sp}}\colon N' \to N$ where $N' = \{B_q \mid B \in N \setminus \{S\}, q \in \{1, 2\}\} \cup \{S\}$ and

$$\mu_{\mathrm{sp}}(B') = \begin{cases} B & \text{if } B' = B_1 \text{ or } B' = B_2 \text{ with } B \in N \\ B' & \text{if } B' = S \end{cases} \quad . \qquad \square$$

[1] Dietze (2018, Sec. 5.1.1) discusses an alternative method to choose p' where for each $B' \in N'$ and $\varrho \in R'_{B'}$ it holds that $p(\mu(\varrho)) = \sum_{\varrho' \in \mu^{-1}(\mu(\varrho))} p'(\varrho)$. In this case, it can be shown that p' is proper, $L(G) = L(G')$, and, for each $t \in \mathrm{T}_\Sigma$, $W_{(G,p)}(t) = W_{(G',p')}(t)$ (cf. Dietze 2018, Thm. 5.1.5). Consequently, (G', p') is a PRTG.

[2] This is not a restriction in expressiveness, because tree automata that allow multiple initial states can be brought into a normal form with just a single initial state. This holds also for weighted tree automata. Also the properties convergence, properness, and consistency are preserved in the weighted case (cf. Maletti and Satta 2009, Thm. 3).

Tie breaking. Let (G, p) be a probabilistic RTG with $G = (N, \Sigma, S, R)$, μ be a splitter for G, $G' = \mu^{-1}(G)$, and $p' = \mathrm{norm}(p \circ \mu)$. Let ϱ in R. There is $\gamma \in \mathbb{R}$ such that, for each $\varrho' \in \mu^{-1}(\varrho)$, we have $p'(\varrho') = p(\varrho) \cdot \gamma$. Also, for each $B \in N$ and $B_1, B_2 \in \mu^{-1}(B)$ we have $\alpha_{(G',p')}(B_1) = \alpha_{(G',p')}(B_2)$ and $\beta_{(G',p')}(B_1) = \beta_{(G',p')}(B_2)$. Recall that after the splitting step, EM training shall be applied. We initialize p_0 with p'. During the expectation step, we compute $c_1 \colon R' \to \mathbb{R}_{\geq 0}$. Due to the preceding observation, we have for $\varrho', \varrho'' \in \mu^{-1}(\varrho)$ that $c_1(\varrho') = c_1(\varrho'')$ and, thus, $p_2(\varrho') = p_2(\varrho'')$. In other words, the different splits of our rule ϱ cannot specialize as intended.

The EM algorithm can escape from this equilibrium if some noise is added to p':

Definition 3.3.4. Let p be a probability assignment for G and let $\alpha \in [0, 1]_{\mathbb{R}}$. We define $\textsc{BreakTies}_\alpha(p)$ to be the probability assignment for G such that

$$\begin{aligned} \textsc{BreakTies}_\alpha(p) &= \mathrm{norm}(w) \\ \text{with} \quad w(\varrho) &= p(\varrho) \cdot r_\varrho \end{aligned}$$

for each $\varrho \in R$ with r_ϱ drawn randomly from the interval $[1 - \alpha, 1 + \alpha]_{\mathbb{R}}$ with values distributed uniformly. □

3.3.2 Efficient refinement of a chart

After we split our grammar G (which we call G^c in the following) and obtained the refined grammar G' (which we call G^f in the following), we execute the EM algorithm. To this end, we need to compute the chart G^f_a for each $a \in \mathcal{A}$ where $c_{\mathcal{A}}(a) > 0$. We do not need to recompute G^f_a from scratch, which might be expensive. One option is to remember the decomposition $D(a)$ for each $a \in \mathcal{A}$ and intersect $D(a)$ and G^f. Here we present an alternative approach that is based on refining already existing charts G^c_a. We construct (a grammar that is isomorphic to) G^f_a via two grammar morphisms $\varphi_a^{G^f}$ and μ_a. These grammar morphisms are chosen such that the diagram in Figure 3.3 commutes, i.e., $\mu(\varphi_a^{G^f}(G^f_a)) = \varphi_a^{G^c}(\mu_a(G^f_a))$.

$$\begin{array}{ccc} G^f_a & \xrightarrow{\varphi_a^{G^f}} & G^f \\ {\scriptstyle \mu_a}\downarrow & & \downarrow{\scriptstyle \mu} \\ G^c_a & \xrightarrow[\varphi_a^{G^c}]{} & G^c \end{array}$$

Figure 3.3: A diagram for chart refinement.

Definition 3.3.5. Let $\mathbb{G} = (G^c, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG where $G^c = (N^c, \Sigma, S^c, R^c)$, $I \subseteq [k]$, $a \in \mathcal{A}_I$, and $G^c_a = (N^c_a, \Sigma, S^c_a, R^c_a)$. Let μ be a splitter for G^c and $G^f =$

$\mu^{-1}(G^c)$. Let $N' = \{(B, q) \mid B \in N_a^c, q \in \mu^{-1}(\varphi_a^{G^c}(B))\}$ and, for every $(B, q) \in N'$, set $\varphi_a^{G^f}(B, q) = q$ and $\mu_a(B, q) = B$. We define $G_a^f = {\mu_a}^{-1}(G_a^c)$. □

Proof for Figure 3.3. Let $(B, q) \in N'$. Then

$$\mu(\varphi_a^{G^f}((B, q))) = \mu(q) = \varphi_a^{G^c}(B) = \varphi_a^{G^c}(\mu_a(B, q)) \ . \qquad \blacksquare$$

3.3.3 Merging

Next we want to describe the reverse step of splitting. Again our formalization is based on a particular grammar morphism.

Definition 3.3.6. Let (G, p) be a probabilistic RTG with $G = (N, \Sigma, S, R)$. A *merger for* G is a surjective mapping $\mu\colon N \to M$ where M is an alphabet and $\mu^{-1}(\mu(S)) = \{S\}$. The *merge of* G *with respect to* μ is $(M, \Sigma, \mu(S), \{\mu(\varrho) \mid \varrho \in R\})$.

We define the probability assignment $\mu(p)$ for $\mu(G)$ such that

$$\mu(p) = \arg\max_{q \in \mathcal{M}(\mu(G))} \mathrm{KL}\left(\lambda d'.\mathrm{P}(d' \mid \mu(G), q) \mid\mid \lambda d'.\mathrm{P}(d' \mid G, p)\right)$$

where

$$\mathrm{P}(d' \mid G, p) = \sum_{d \in \mu^{-1}(d') \cap (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \ .$$

for each $d' \in \{\mu(d) \mid d \in (\mathrm{T}_R)_S\}$. □

The next theorem provides a way to compute $\mu(\varrho)$.

Theorem 3.3.7. Let (G, p) be a probabilistic RTG with $G = (N, \Sigma, R, S)$, μ be a merger for G, and $\mu(G) = (N', \Sigma, S', R')$. Then, for every ϱ' in R' of the form $B' \to \sigma(B'_1, \ldots, B'_n)$, we have

$$(\mu(p))(\varrho') = \frac{\sum_{\varrho \in R\colon \mu(\varrho) = \varrho'} \alpha_{(G,p)}(\varrho) \cdot p(\varrho) \cdot \beta_{(G,p)}(\varrho)}{\sum_{B \in N\colon \mu(B) = B'} \alpha_{(G,p)}(B) \cdot \beta_{(G,p)}(B)} \ . \qquad \square$$

Proof. Let ϱ' in $R' = \mu(R)$ be of the form $B' \to \sigma(B'_1, \ldots, B'_k)$. Then

$$\begin{aligned}
&\frac{\sum_{\varrho \in R\colon \mu(\varrho) = \varrho'} \alpha_{(G,p)}(\varrho) \cdot p(\varrho) \cdot \beta_{(G,p)}(\varrho)}{\sum_{B \in N\colon \mu(B) = B'} \alpha_{(G,p)}(B) \cdot \beta_{(G,p)}(B)} \\
&= \frac{\sum_{\varrho \in R\colon \mu(\varrho) = \varrho'} \alpha_{(G,p)}(\varrho) \cdot p(\varrho) \cdot \beta_{(G,p)}(\varrho)/\beta_{(G,p)}(S)}{\sum_{B \in N\colon \mu(B) = B'} \alpha_{(G,p)}(B) \cdot \beta_{(G,p)}(B)/\beta_{(G,p)}(S)} \\
&= \frac{\sum_{\varrho \in R\colon \mu(\varrho) = \varrho'} \mathbb{E}_{\lambda d.\mathrm{P}(d|G,p)}\left[\mathrm{occ}_\varrho\right]}{\sum_{B \in N\colon \mu(B) = B'} \mathbb{E}_{\lambda d.\mathrm{P}(d|G,p)}\left[\mathrm{occ}_B\right]} && \text{(Lemma 3.2.4)} \\
&= \frac{\sum_{\varrho \in R\colon \mu(\varrho) = \varrho'} \sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_\varrho(d)}{\sum_{B \in N\colon \mu(B) = B'} \sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_B(d)}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \sum_{\varrho \in R\colon \mu(\varrho) = \varrho'} \mathrm{occ}_{\varrho}(d)}{\sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \sum_{B \in N\colon \mu(B) = B'} \mathrm{occ}_{B}(d)} \\
&= \frac{\sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_{\varrho'}(\mu(d))}{\sum_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_{\mu(B')}(\mu(d))} \\
&= \frac{\sum_{d' \in (\mathrm{T}_{R'})_{S'}} \sum_{d \in \mu^{-1}(d)} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_{\varrho'}(d')}{\sum_{d' \in (\mathrm{T}_{R'})_{S'}} \sum_{d \in \mu^{-1}(d)} \mathrm{P}(d \mid G, p) \cdot \mathrm{occ}_{\mu(B')}(d')} \\
&= \frac{\sum_{d' \in (\mathrm{T}_{R'})_{S'}} \mathrm{P}(d' \mid G, p) \cdot \mathrm{occ}_{\varrho'}(d')}{\sum_{d' \in (\mathrm{T}_{R'})_{S'}} \mathrm{P}(d' \mid G, p) \cdot \mathrm{occ}_{\mu(B')}(d')} \\
&= \frac{\mathbb{E}_{\lambda d'.\mathrm{P}(d' \mid G, p)} \left[\mathrm{occ}_{\varrho'}\right]}{\mathbb{E}_{\lambda d'.\mathrm{P}(d' \mid G, p)} \left[\mathrm{occ}_{B'}\right]} \\
&= \arg\max_{q \in \mathcal{M}(\mu(G))} \mathrm{KL}\left(\lambda d'.\mathrm{P}(d' \mid \mu(G), q) \;||\; \lambda d'.\mathrm{P}(d' \mid G, p)\right) \quad \text{(Lemma 3.1.13)} \\
&= \mu(p)(\varrho') \qquad \blacksquare
\end{aligned}
$$

Now that we have presented the general methodology for merging, we turn to the question of how to select an appropriate merger μ. Clearly, in the context of some IRTG $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ where $G = (N, \Sigma, S, R)$, we need to make sure that μ does not map nonterminals from N with different sorts to the same nonterminal in M. Otherwise, we loose the property that $\mu(G)$ is compatible to $\mathcal{A}_1$, …, $\mathcal{A}_k$. Formally, for each $B, B' \in N$, we require that $\mu(B) = \mu(B')$ implies $\mathrm{sort}_N(B) = \mathrm{sort}_N(B')$.

Fortunately, in the split/merge algorithm we only want to undo splits of some nonterminal from a coarse grammar. Hence, this is not a problem. However, another problem arises: we need to select which splits are useful. For now we abstract from this problem by a function Δ that maps pairs of nonterminals to the cost of keeping these nonterminals split. A value η defines the maximum cost that we are willing to accept. Next, we give the formal definition of such a merger.

Definition 3.3.8. Let (G, p) be a probabilistic RTG, let $G = (N, \Sigma, S, R)$, and let $\mu_{\mathrm{sp}}\colon N' \to N$ be the 2-splitter of G. Let

$$\Delta\colon \{(A, B) \in N' \times N' \mid A \neq B, \mu_{\mathrm{sp}}(A) = \mu_{\mathrm{sp}}(B)\} \to \mathbb{R}$$

and let $\eta \in \mathbb{R}$. The *Δ-merger of $\mu_{\mathrm{sp}}(G)$* is a mapping $\mu_\Delta\colon N' \to M$ where

$$\mu_\Delta(B') = \begin{cases} B & \text{if } B' = B_q \text{ for } B \in N,\ q \in \{1, 2\}, \text{ and } \Delta(B_1, B_2) > \eta \\ B' & \text{otherwise.} \end{cases}$$

and M is the largest subset of $N \cup N'$ such that μ_Δ is surjective. □

Likelihood-loss approximation. Petrov et al. (2006) propose a Δ-merger where the cost that Δ assigns to merging two refined nonterminals B_1 and B_2 is an

approximation of the quotient of the likelihood of the training corpus after and before merging B_1 and B_2. Thus, if merging does not reduce the likelihood too much, then the cost of keeping the split are high. We formalize this function Δ for the IRTG framework.

Let $\mathbb{G} = (G, \mathcal{A}_1, \dots, \mathcal{A}_k)$ be an IRTG, let $\mu_{\mathrm{sp}}(G) = G^f = (N^f, \Sigma, S^f, R^f)$, let $p \in \mathcal{M}(G^f)$, let $a \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$, let $G_a^f = (N_a^f, \Sigma, S_a^f, R_a^f)$, and let $p_a^f = p \circ \varphi_a^{G^f}$. Let $B_1, B_2 \in N^f$, $B_1' \in \varphi_a^{G^f\,-1}(B_1)$ and $B_2' \in \varphi_a^{G^f\,-1}(B_1)$ be such that $(\mu_{\mathrm{sp}})_a(B_1') = (\mu_{\mathrm{sp}})_a(B_2')$, where $(\mu_{\mathrm{sp}})_a$ is the grammar morphism from G_a^f to G_a defined as in Definition 3.3.5. Intuitively, B_1' and B_2' are refinements of a nonterminal B' in the coarse chart G_a. We consider what happens if we merge just B_1' and B_2' back to B' (while keeping all other occurrences of B_1 and B_2 in charts unmerged). We assign B' an inside and an outside weight by setting

$$\begin{aligned}
\alpha_{(G_a^f, p_a^f)}(B') &= \alpha_{(G_a^f, p_a^f)}(B_1') + \alpha_{(G_a^f, p_a^f)}(B_2') \\
\beta_{(G_a^f, p_a^f)}(B') &= p_1 \cdot \beta_{(G_a^f, p_a^f)}(B_1') + p_2 \cdot \beta_{(G_a^f, p_a^f)}(B_2')
\end{aligned}$$

where p_1 and p_2 are so-called *merge factors* that describe the ratio of the expected frequencies of B_1' and B_2':

$$p_i = \frac{\mathbb{E}_{\lambda d.\mathrm{P}(d|G_a^f, p_a^f)}\left[\mathrm{occ}_{B_i'}\right]}{\mathbb{E}_{\lambda d.\mathrm{P}(d|G_a^f, p_a^f)}\left[\mathrm{occ}_{B_1'}\right] + \mathbb{E}_{\lambda d.\mathrm{P}(d|G_a^f, p_a^f)}\left[\mathrm{occ}_{B_2'}\right]} \ .$$

We call the merged chart $\mu_{(B_1', B_2')}(G_a^f) = G_a^\mu = (N_a^\mu, \Sigma, S_a^\mu, R_a^\mu)$ and $p_a^\mu = \mu_{(B_1', B_2')}(p_a^f)$. Applying this merge has the following influence on the probability of a:

$$\begin{aligned}
&\sum_{d \in (\mathrm{T}_{R_a^\mu})_{S_a^\mu}} W_{(G_a^\mu, p_a^\mu)}(d) \\
&= \sum_{d \in (\mathrm{T}_{R_a^f})_{S_a^f} :\, \mathrm{occ}_{B_i'}(d) = 0} W_{(G_a^f, p_a^f)}(d) + \sum_{\substack{d' \in \{\mu_{(B_1', B_2')}(d) | \\ d \in (\mathrm{T}_{R_a^f})_{S_a^f} :\, \mathrm{occ}_{B_i'}(d) > 0\}}} W_{(G_a^\mu, p_a^\mu)}(d') \\
&= \sum_{d \in (\mathrm{T}_{R_a^f})_{S_a^f}} W_{(G_a^f, p_a^f)}(d) - \sum_{\substack{d \in (\mathrm{T}_{R_a^f})_{S_a^f} : \\ \mathrm{occ}_{B_i'}(d) > 0}} W_{(G_a^f, p_a^f)}(d) + \sum_{\substack{d' \in \{\mu_{(B_1', B_2')}(d) | \\ d \in (\mathrm{T}_{R_a^f})_{S_a^f} :\, \mathrm{occ}_{B_i'}(d) > 0\}}} W_{(G_a^\mu, p_a^\mu)}(d') \\
&= \beta_{(G_a^f, p_a^f)}(S_a^f) - \left(\sum_{i \in \{1,2\}} \alpha_{(G_a^f, p_a^f)}(B_i') \cdot \beta_{(G_a^f, p_a^f)}(B_i') \right) + \alpha_{(G_a^f, p_a^f)}(B') \cdot \beta_{(G_a^f, p_a^f)}(B')
\end{aligned} \tag{3.11}$$

For the last equality to hold, we need that at most one of B_1' and B_2' occurs in

any derivation tree of G_a^f. Otherwise, certain derivations are accounted for multiple times.[3] We notice that all quantities in (3.11) can be efficiently computed.

Using the above formula, we can finally define the function Δ that approximates the quotient of likelihood after and before merging B_1 and B_2 by accumulating over likelihood quotients for occurrences of B_1 and B_2 in charts:

$$\Delta(B_1, B_2) = \prod_{\substack{a \in \mathcal{A} \\ (B_1', B_2') \text{ in } N_a^f}} \left(\frac{\sum_{d \in (\mathrm{T}_{R_a^\mu})_{S_a^\mu}} W_{G_a^\mu, p_a^\mu}(d)}{\sum_{d \in (\mathrm{T}_{R_a^f})_{S_a^f}} W_{(G_a^f, p_a^f)}(d)} \right)^{c_{\mathcal{A}}(a)}$$

In the above formula, the numerator shall express the probability of a after merging and the denominator expresses the probability of a before merging.

After $\Delta(B_1, B_2)$ has been computed for each relevant pair of nonterminals (B_1, B_2) of G^f, η is chosen such that a fixed percentage (e.g., 50%) of splits is reversed.

3.3.4 Smoothing

Let us reconsider a problem we already addressed in Section 3.2.4. The maximum likelihood estimate $p^* = \arg\max_{p \in \mathcal{M}(G)} \sum_{a \in \mathcal{A}} c_{\mathcal{A}}(a) \cdot \log(\mathrm{P}(a \mid G, p))$ is prone to *overfitting*, i.e., the distribution $\lambda a.\mathrm{P}(a \mid G, p^*)$ is likely to assign little or even no probability mass to elements $a \in \mathcal{A}$ where $c_{\mathcal{A}}(a) = 0$. However, when the grammar is applied during testing, it should *generalize*, i.e., be able to handle unseen domain objects $a \in \mathcal{A}$.

The robustness (see also p. 126) of a grammar that is obtained by splitting nonterminals repeatedly can be increased by a particular variant of *smoothing*. It is imposed that the weights assigned to different refinements of the same rule shall be similar to a certain degree. Although it might be possible to encode such a restriction into an elaborated prior, we follow Petrov et al. (2006) by applying the transformation $\textsc{smooth}_\gamma$ to the rule weights.

Definition 3.3.9. Let (G, p) be a probabilistic RTG, μ be a grammar morphism from G to RTG G', and $\gamma \in [0, 1]_{\mathbb{R}}$. $\textsc{smooth}_\gamma(G, p)$ is the probability assignment for

[3] In many relevant cases this requirement is met: If a nonterminal of the decomposition $D(a)$ does occur at most once in a derivation, then this also holds for the chart $G_a = G \cap D(a)$ and its refinements G_a^f and G_a^μ. Decompositions $D(a)$ can often be designed to use substructures of a as nonterminals, where rules decompose a substructure a' into *smaller*, *non-overlapping* substructures $a_1', \ldots, a_k'$. In consequence, in a derivation of $D(a)$ each nonterminal occurs at most once. A counter example to this are cycles in $D(a)$, which, e.g., occur due to chain rules $a' \to \gamma(a')$ where $\gamma^{\mathcal{A}}(x) = x$.

G where, for each rule ϱ' of the form $A' \to \sigma(A'_1, \ldots, A'_k)$, we have

$$(\textsc{smooth}_\gamma(G,p))(\varrho') = \frac{w(\varrho')}{\sum_{\varrho'' \in R_{A'}} w(\varrho'')}$$

$$\text{with} \quad w(\varrho') = p(\varrho') \cdot (1-\gamma) + \left(\sum_{\varrho' \in \mu^{-1}(\mu(\varrho))} p(\varrho') \right) \cdot \gamma \ . \qquad \square$$

3.3.5 Split/Merge Cycles

A complete split merge cycle is formalized as Algorithm 3.3.1. Multiple of these cycles are applied until the grammar satisfies the desired performance requirements. As we will show Chapter 7, after a certain number of cycles the accuracy on the grammar on a held-out set does not improve any more. However, each additional cycle increases the number of nonterminals, rules, and rule weights, which slows down the algorithms that depend on the grammar.

Algorithm 3.3.1 Split/merge cycle

Input: probabilistic IRTG $(\mathbb{G}, p)$ where $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$
corpus $c_\mathcal{A} \colon \mathcal{A} \to \mathbb{R}_{\geq 0}$
Output: probabilistic IRTG $(\mathbb{G}', p')$ with $\mathbb{G} = (G', \mathcal{A}_1, \ldots, \mathcal{A}_k)$

1: $(G^f, p^f) \leftarrow (\mu_{\text{sp}}^{-1}(G), \text{norm}(p \circ \mu_{\text{sp}}))$
2: $p^f \leftarrow \textsc{BreakTies}(p^f)$
3: $p^f \leftarrow \text{EM-}\textsc{Training}((G^f, \mathcal{A}_1, \ldots, \mathcal{A}_k), p^f, c_\mathcal{A})$
4: $(G', p') \leftarrow (\mu_\Delta(G^f), \mu_\Delta(p^f))$
5: $p' \leftarrow \text{EM-}\textsc{Training}((G', \mathcal{A}_1, \ldots, \mathcal{A}_k), p', c_\mathcal{A})$
6: $p' \leftarrow \textsc{smooth}_\gamma(G', p')$
7: $p' \leftarrow \text{EM-}\textsc{Training}((G', \mathcal{A}_1, \ldots, \mathcal{A}_k), p', c_\mathcal{A})$
8: **output** $(G', \mathcal{A}_1, \ldots, \mathcal{A}_k), p'$

Now that we have presented a training scheme for IRTG, we complete this chapter by considering ways to use split/merge-refined grammars for parsing.

3.4 Parsing objectives for probabilistic IRTG

In this section we consider different ways to solve the parsing problem for probabilistic IRTG. A first step, that we ignore at the time being, is the computation of the chart $G_{a'}$ for the input a' because it depends on the specific algebras in use. Therefore, we concentrate on solving parsing problems for convergent weighted RTG, which turn out

to be very hard. A substantial body of work exists for closely related formalism with hard parsing problems, e.g., for tree substitution grammars (cf. Bod 1995; Sima'an 2002; Sangati and Zuidema 2011) and for PCFG-LA (Matsuzaki, Miyao, and Tsujii 2005; Petrov et al. 2006; Petrov and Klein 2007).

Throughout this section, let $\mathbb{G} = (G, \mathcal{A}_1, \ldots, \mathcal{A}_k)$ be an IRTG and p be a probability assignment for G. Moreover let $I \subset [k]$, $J = [k] \setminus I$, $a' \in \mathcal{A}_I$, and $a'' \in \mathcal{A}_J$.

3.4.1 NP-hardness of PRTG parsing

Sima'an (2002) showed by a reduction of 3SAT[4] that various parsing problems for *stochastic tree substitution grammars* (STSG) are NP-hard. In particular, finding the *most probable parse*, i.e., given an STSG G and a string w, we look for the most probable tree with yield w, is NP-hard. Matsuzaki, Miyao, and Tsujii (2005) state that this problem can be reduced to finding the most probable parse of some PCFG-LA G' for w. In consequence, they obtain NP-hardness for PCFG-LA parsing. A reduction of 3SAT to finding the most probable tree of a PRTG can be made similarly.[5]

Theorem 3.4.1. Let $G = (G, p)$ be a convergent weighted RTG. Computing the most probable tree for (G, p), i.e.,

$$\arg\max_{t \in \mathrm{T}_\Sigma} \mathrm{P}(t \mid G, p) \ , \tag{3.1}$$

is NP-hard. □

As Maletti and Satta (2009) note, NP-hardness also follows from the NP-hardness of finding the *most probable string* generated by a probabilistic string automaton (Casacuberta and de la Higuera 2000), which can be simulated by probabilistic tree automata or, equivalently, an PRTG. A related line of research by de la Higuera and Oncina is concerned with developing algorithms to actually find the most probable string, here also called *consensus string*: de la Higuera and Oncina (2013a) show that the consensus string can be of length exceeding any polynomial bound in the number of states.[6] Later, de la Higuera and Oncina (2013b) provide an upper bound on the weight of a string given its length and develop an algorithm that computes the consensus string in time polynomial to the inverse of its probability. Again, similar properties and algorithms can be derived for probabilistic tree automata (Meinert

[4] 3SAT is the satisfiability problem for propositional formulas in conjunctive normal form where each clause contains at most three literals.

[5] A direct proof has been given by Meinert (2019) under my supervision.

[6] De la Higuera and Oncina (2013a) falsely state that the length of the consensus string can be exponential in the number of states. For this, the authors refer to de la Higuera (1997). The latter applies the same construction but it is only claimed and proved that $\mathcal{O}(n^{\log_2 n})$ is a lower bound on the length of the consensus string where n is the number of states of the automaton. The function $f(n) = n^{\log_2(n)} = 2^{(\log_2(n)^2)} \ll 2^n$ grows faster than polynomial but not exponential.

2019). The experiments by Meinert (2019) indicate that the runtime of his most-probable tree algorithm is too high to be of practical interest. Nevertheless, we want to remark that a tight upper bound (independent of the probability of the resulting tree) for the complexity of the most probable tree problem for PRTG would be at least interesting from a theoretical viewpoint.

How does the NP-hardness of PRTG parsing relate to probabilistic IRTG parsing? Theorem 3.4.1 and the knowledge that the size of the most-probable tree cannot be bounded by a polynomial indicate that there is no tractable way of solving the parsing problem for probabilistic IRTG, as stated in Equation (3.4). This is because the reductions presented to compute $\mathrm{P}(a'' \mid a', \mathbb{G}, p)$ rely on computing Equation (3.1) for $(G_{a'}, p \circ \varphi_{a'})$. Hence, we explore tractable simplifications of the parsing problem following the literature.

3.4.2 Viterbi parsing

The goal of *Viterbi parsing*[7] is computing the *Viterbi parse tree*, i.e., the tree $\hat{t}$ in T_Σ that corresponds to the *most probable derivation* of $(\mathbb{G}, p)$ for a':

$$\hat{t} = \left[\!\!\left[\arg\max_{d \in (\mathrm{T}_R)_S} \mathrm{P}(d \mid a', \mathbb{G}, p) \right]\!\!\right]_S^{\Pi} \quad .$$

We refer to $\hat{t}$'s interpretation a'' in $\mathcal{A}^J$ as *Viterbi parse*:

$$a'' = [\![\hat{t}]\!]^{\mathcal{A}_J} \quad .$$

Finding the most probable derivation can be done in polynomial time once the chart of a' has been computed. To this end, Knuth's generalization of the Dijkstra algorithm (Knuth 1977) can be used.

If each tree $t \in \mathrm{T}_\Sigma$ has at most one derivation or the probability of one derivation dominates the probability mass assigned to t, then the Viterbi parsing objective is a reasonable supplement for exact PRTG parsing. However, empirical data suggests that the premise of this implication does not hold: Bod (1995) reports an experiment with an STSG and a real-world corpus where only in 68% of the cases, the most probable derivation belongs to the most probable parse. At the same time, he finds the most probable parse to match the desired output more often. Concerning probabilistic string automata, de la Higuera and Oncina (2013b) found that in 63% of the cases the most probable string deviates from the string corresponding to the best run on a synthetic benchmark with highly ambiguous grammars. Also, Meinert (2019) reports deviations between the most probable parse and the Viterbi parse for PRTG.

[7] Named after Viterbi (1967), who developed an algorithm to compute the most probable sequence of hidden states given a hidden Markov model and a sequence of observed symbols.

3.4.3 Sampling

An approach that was proposed by Bod (1995) for STSG is *sampling* from $\lambda d.\mathrm{P}(d \mid G_{a'}, p \circ \varphi_{a'})$. To this end, $(G_{a'}, p \circ \varphi_{a'})$ is normalized to a PRTG $(G_{a'}, p')$ (cf. Theorem 3.2.5). Subsequently, a derivation can be sampled by starting from the start symbol and recursively selecting a rule ϱ with probability $p'(\varrho)$ for each nonterminal.

In order to approximate the most probable parse, sufficiently many derivations (say n) are sampled and each derivation d is interpreted to $t = [\![d]\!]_S^{\Pi}$. The tree t that is obtained most often is the result. As Bod (1995) elaborates, by the law of large numbers, in the limit for $n \to \infty$ we obtain the most probable tree. Also upper bounds on the error can be given depending on n. We do not further investigate sampling in this work.

3.4.4 n-best parsing

Instead of considering just the Viterbi parse, it may be helpful to compute the n most probable derivations and their corresponding parse trees. To this end, different algorithms have been compared by L. Huang and Chiang (2005). For a fixed n the overhead over Viterbi parsing is polynomial and, for small values of n, in practise often negligible. Specifically, the objective is to compute

$$[d_1, \ldots, d_\kappa]$$

such that $\kappa \leq n$ and

- for each $i \in [\kappa]$, $d_i \in (\mathrm{T}_R)_S \setminus \{d_1, \ldots, d_{i-1}\}$ and $\left[\!\left[[\![d_i]\!]_S^{\Pi} \right]\!\right]^{\mathcal{A}_I} = a'$,
- $\mathrm{P}(d_i \mid a', \mathbb{G}, p) \geq \mathrm{P}(d_{i+1} \mid a', \mathbb{G}, p)$ for each $i \in [\kappa - 1]$,
- there is no $d \in (\mathrm{T}_R)_S \setminus \{d_1, \ldots, d_\kappa\}$ with $\left[\!\left[[\![d]\!]_S^{\Pi} \right]\!\right]^{\mathcal{A}_I} = a'$ and $\mathrm{P}(d \mid a', \mathbb{G}, p) > \mathrm{P}(d_\kappa \mid a', \mathbb{G}, p)$, and
- if $\kappa < n$, then there is no $d \in (\mathrm{T}_R)_S \setminus \{d_1, \ldots, d_\kappa\}$ with $\left[\!\left[[\![d]\!]_S^{\Pi} \right]\!\right]^{\mathcal{A}_I} = a'$.

Notably, two distinct derivation d_i and d_j may be such that $[\![d_i]\!]_S^{\Pi} = [\![d_j]\!]_S^{\Pi}$. One can account for this phenomenon by a technique called *crunching* (cf. May and Knight 2006). Here, one computes a list $[t_1, \ldots, t_{\kappa'}]$ of $\kappa' \leq \kappa$ distinct parse trees in T_Σ. For each $i \in [\kappa']$, there is a derivation $d_{i'}$ such that $[\![d_{i'}]\!]_S^{\Pi} = t_i$ and the weight w_i of t_i is computed such that

$$w_i = \sum_{i' \in [\kappa] \colon [\![d_{i'}]\!]_S^{\Pi} = t_i} \mathrm{P}(d_{i'} \mid a', \mathbb{G}, p) \ .$$

The trees are ordered such that $w_i \geq w_{i+1}$ for each $i \in [\kappa' - 1]$.

Alternatively, Björklund, Drewes, and Zechner (2015) present an algorithm that computes a list of (up to) n best distinct trees

$$[t_1, \ldots, t_\kappa]$$

in polynomial time given a probabilistic tree automaton. The trees are ordered in accordance to the probability w_i of their best derivation:

$$w_i = \max_{d' \in (\mathrm{T}_R)_S \colon [\![d']\!]_S^{\Pi} = t_i} \mathrm{P}(d' \mid a', \mathbb{G}, p) \ .$$

3.4.5 Reranking n-best derivation trees

If we consider a grammar G^f that was refined using the split/merge algorithm, then the *spurious ambiguity*, i.e., the number of derivations for a fixed tree, is typically very high. This has two negative consequences: (i) The computation of the chart $G^f_{a'}$ requires a lot of redundant work. (ii) The deviation between the most probable derivation and the most probable tree is likely to be larger as with the baseline PRTG.

In order to keep the parsing problem tractable, one can combine polynomial n-best parsing with the coarse grammar G^c with a polynomial reranking approach. In principal, n-best derivations of the coarse grammar are computed and the corresponding list of derivations or trees is rescored according to the refined grammar. Formally, we compute $a'' = \left[\!\left[[\![d_{\hat{i}}]\!]_S^{\Pi} \right]\!\right]_{\mathcal{A}_J}$ with

$$\hat{i} = \arg\max_{i \in [\kappa]} \sum_{d' \in \mu_{a'}{}^{-1}(d_i)} \mathrm{P}(d' \mid (G^f, \mathcal{A}_1, \ldots, \mathcal{A}_k), p^f)$$

where $[d_1, \ldots, d_\kappa]$ are the n-best derivations for $\lambda d.\mathrm{P}(d \mid a', (G^c, \mathcal{A}_1, \ldots, \mathcal{A}_k), p^c)$.

Alternatively, one can choose

$$\hat{i} = \arg\max_{i \in [\kappa]} \mathrm{P}([\![d_i]\!]_S^{\Pi} \mid a', (G^f, \mathcal{A}_1, \ldots, \mathcal{A}_k), p^f) \ ,$$

which requires, for each $i \in [\kappa]$, the intersection of $G^f_{a'}$ and $[\![d_i]\!]_S^{\Pi}$ and the computation of the inside weight of the initial nonterminal of the resulting grammar.[8]

Other prominent versions of reranking add a discriminative model on top of a grammar (e.g., Charniak and Johnson 2005; Choe and Charniak 2016). A general problem of n-best reranking is that the best parses according to the refined grammar (or discriminative model) may not be among the n-best ones of the baseline grammar (see e.g., Lin et al. 2019).

[8] Recall that for each each tree t in T_Σ the language $\{t\}$ is regular and that RTG are closed under intersection.

3.4.6 Projection-based parsing strategies

Variational. Matsuzaki, Miyao, and Tsujii (2005) propose an alternative parsing objective, where a new weight assignment q for $G^c_{a'}$ is computed based on $(G^f_{a'}, p^f \circ \varphi^{G^f}_{a'})$ such that the KL-divergence of $\lambda d.\mathrm{P}(d \mid G^c_{a'}, q)$ to $\lambda d.\mathrm{P}(d \mid a', (G^f, \mathcal{A}_1, \ldots, \mathcal{A}_k), p^f)$ is minimized. Precisely, $q = \mu_{a'}(p^f \circ \varphi^{G^f}_{a'})$ (see Definition 3.3.6). Subsequently the parse tree a'' corresponding to the Viterbi derivation tree is computed using q, i.e.,

$$a'' = \left[\!\!\left[\left[\!\!\left[\hat{d} \right]\!\!\right]_S^{\Pi} \right]\!\!\right]^{\mathcal{A}_J} \qquad \text{with} \qquad \hat{d} = \operatorname*{arg\,max}_{d \in (\mathrm{T}_{R^c})_{S^c_{a'}}} \mathrm{P}(d \mid G^c_{a'}, q) \ .$$

Max-rule-product. An empirically superior way to define q called *max-rule-product* is due to Petrov and Klein (2007). They intent to optimize for "the tree with greatest chance of having all rules correct, under the (incorrect) assumption that the rules' correctness is independent." To this end, each rule is assigned the sum of the expected frequencies of its refinements, i.e.,

$$q(\varrho) = \sum_{\varrho' \in {\mu_{a'}}^{-1}(\varrho)} \frac{\alpha_{(G^f_{a'}, p^f \circ \varphi^{G^f}_{a'})}(\varrho') \cdot (p^f \circ \varphi^{G^f}_{a'})(\varrho') \cdot \beta_{(G^f_{a'}, p^f \circ \varphi^{G^f}_{a'})}(\varrho')}{\beta_{(G^f_{a'}, p^f \circ \varphi^{G^f}_{a'})}(S^f_{a'})} \ .$$

Hence, the value $q(\varrho)$ does not have to be in the interval $[0,1]_{\mathbb{R}}$ and max-rule-product is in theory a potentially non-monotonic[9] and, thus, ill-defined objective (cf. Appendix A.1). We do not find this theoretical problem to occur in our experiments.

Product of projections. An orthogonal strategy to the variational and the max-rule-product objective was proposed by Petrov (2010). He suggests that grammars obtained using the split/merge algorithm with different random seeds specialize differently. For a sequence of different refined grammars (G^f_1, p^f_1), …, (G^f_n, p^f_n), the projected weight assignments $q_1, \ldots, q_n$ for $G^c_{a'}$ are computed according to the variational or max-rule-product strategy. Afterward, a new weight assignment $\hat{q}$ is obtained where $\hat{q}(\varrho) = \prod_{i=1}^n q_i(\varrho)$ for each rule ϱ in G^c_a. Intuitively, the product shall combine the "wisdom" of the different refined grammars by allowing each to veto a rule.

Max-rule-sum. In this strategy, the same weight assignment as in max-rule-product is used. However, the weight of a derivation in the coarse chart is defined to be the sum of rule weights rather than the product. This objective optimizes the expected recall of correct rules, i.e., it favors parses that contain many rules with

[9] When the Dijkstra algorithm or one of its generalizations is applied, then the weight structure must satisfy monotonicity in order for the algorithm to be correct: extending any path must always result in a worse or equal score.

high expected frequency. Obviously, this objective is non-monotonic as well, because adding more rules will always increase the weight of a derivation. When Petrov and Klein (2007) apply this strategy during PCFG-LA parsing, the only critical situations are unary chain rules of the form $A \to \gamma(B)$ because other rules generate non-empty parts of the input a'. Therefore, ibid. impose a limit on the number of chain rules per span they consider during parsing.

Maximum constituents. This strategy is arguably the oldest (Goodman 1996a; Goodman 1996b) of the projection-based ones and was proposed for PCFG and STSG. The objective has similarities to max-rule-sum but the goal is to maximize the expected number of correct *constituents*:

$$\arg\max_{t \in \mathrm{T}_\Sigma} \sum_{c \in \mathrm{LC}(t)} \mathbb{E}_{\lambda t'.\mathrm{P}(t'|a',G,p)} \left[\lambda t'. \begin{cases} 1 & \text{if } c \in \mathrm{LC}(t') \\ 0 & \text{otherwise} \end{cases} \right] \tag{3.12}$$

where $\mathrm{LC}(t)$ is the set of labeled constituents of t. A *labeled constituent* is a pair of the label $t(w)$ at a tree position w and the substructure of a' spanned by w, i.e.,

$$\mathrm{LC}(t) = \{(t(w), \mathrm{span}(w, t, a')) \mid w \in \mathrm{pos}(t)\} \ .$$

We assume that $\mathrm{span}(w, t, a')$ associates a particular substructure $\bar{a}$ of a' to each subtree $t|_w$. For instance, if a' is a sentence, then $\mathrm{span}(w, t, a')$ may be the set of all sentence positions which are covered by the subtree $t|_w$.

To make the computation of Equation (3.12) feasible, two additional assumptions are required: (i) The set of considered substructures of a' shall be finite. (ii) For each nonterminal B of the chart $G_{a'}$, there shall be a unique substructure $\bar{a}_B$ of a' such that for each $d \in (\mathrm{T}_R)_B$ we have $\left[\!\left[[\![d]\!]^{II} \right]\!\right]^{\mathcal{A}_I} = \bar{a}_B$. This allows us to construct a new grammar $G' = (N', \Sigma, a', R')$ where

- $N' = \{\bar{a} \mid \bar{a} \text{ is substructure of } a'\}$ and
- $R' = \{\bar{a} \to \sigma(\bar{a}_1, \ldots, \bar{a}_k) \mid k \in \mathbb{N}, \sigma \in \Sigma_k, \bar{a} = \sigma^{\mathcal{A}_I}(\bar{a}_1, \ldots, \bar{a}_k)\}$.

Next, a weight assignment q for G' is defined using the chart $G_{a'} = (N, \Sigma, S, R)$ with probability assignment $p' = p \circ \varphi^G_a$:

$$\begin{aligned} q(\bar{a} \to \sigma(\bar{a}_1, \ldots, \bar{a}_k)) &= \mathbb{E}_{\lambda t'.\mathrm{P}(t'|a',G,p)} \left[\lambda t'. \begin{cases} 1 & \text{if } (\sigma, \bar{a}) \in \mathrm{LC}(t') \\ 0 & \text{otherwise} \end{cases} \right] \\ &= \sum_{\substack{B \in N \colon \bar{a}_B = \bar{a} \\ \varrho = B \to \sigma(B_1, \ldots, B_k) \in R}} \frac{\alpha_{(G_{a'}, p')}(B) \cdot p'(\varrho) \cdot \prod_{i \in [k]} \beta_{(G_{a'}, p')} B_i}{\beta_{(G_{a'}, p')}(S)} \end{aligned}$$

Finally, Equation (3.12) can be approximately solved by searching the derivation tree of (G', q) with the maximum weight using Knuth's algorithm (1977) with the bimonoid $(\mathbb{R}_{\geq 0}, \max, +, 0, 0)$. As this bimonoid is non-monotonic, the maximum height of trees needs to be restricted to ensure termination.

A notable difference of max-rule-sum to the maximum constituents objective is that weights are not projected from the chart of a given fine grammar to that of a given coarse one. Instead, the coarsest grammar G' that still just recognizes a' is constructed, i.e., the state behaviour of G is ignored and arbitrary trees in $\{t \in \mathrm{T}_\Sigma \mid [\![t]\!]^{\mathcal{A}_I} = a'\}$ can be generated.

4 Grammar formalisms for discontinuous/non-projective parsing

This chapter is dedicated to the presentation of three grammar formalisms which we later use for practical parsing experiments. In our presentation each formalism is represented by the combination of an RTG with one or several algebras, i.e., an IRTG.

First we recall *linear context-free rewriting systems* (LCFRS, Vijay-Shanker, Weir, and Joshi 1987) for which the characterization by some kind of initial algebra semantics is usual unless they are presented as simple range concatenation grammars (Boullier 1998).

Afterward, we present a variant of *simple definite clause programs* (sDCP, Deransart and Małuszyński 1985). Instances of this formalism can be viewed as logic programs over unranked trees, where free variables are allowed to occur in the antecedents. Despite their strong relation to *attribute grammars* (Knuth 1968) already indicates a context-free backbone structure, an algebraic view on this formalism is hindered by *inherited attributes* at first glance. However, using second-order variables we are able to eliminate free variables (or inherited attributes).

Lastly, we link LCFRS and sDCP to so-called LCFRS/sDCP hybrid grammars (Nederhof and Vogler 2014; Gebhardt, Nederhof, and Vogler 2017). Here, an LCFRS and an sDCP are linked to share the same context-free backbone structure. Additionally, an alignment between symbols generated by the LCFRS and the symbols generated by the sDCP is established. In order to do this in a way compatible with initial algebra semantics we introduce a novel alignment algebra that keeps track of string positions and tree positions.

4.1 Linear context-free rewriting systems

In order to describe LCFRS, we first introduce a class of algebras which have tuples of strings as domain. Each operation of this algebra receives multiple such tuples of strings as input. It forms a new tuple of strings by concatenating the components of

the input tuples and potentially new terminal symbols. In doing so, each component in each input tuple is used exactly once. As we want to deal with tuples of different arity each operation is assigned a sort from $\mathbb{N}^* \times \mathbb{N}$ which indicates the arities of the input tuples and the output tuple.

Definition 4.1.1. Let Σ be an $\mathbb{N}^* \times \mathbb{N}$-sorted alphabet and Δ be an alphabet. A (Σ, Δ)-*LCFRS algebra* is a Σ-algebra $\mathcal{A} = (\mathcal{A}, \cdot^{\mathcal{A}})$ where $\mathcal{A}_n = (\Delta^*)^n$ and operations satisfy the following: For each $n \in \mathbb{N}$, $k_0, \ldots, k_n \in \mathbb{N}$, and $\sigma \in \Sigma_{(k_1 \ldots k_n, k_0)}$ there are $w_1, \ldots, w_{k_0} \in (\Delta \cup \bar{X})^*$ where $\bar{X} = \{x_1^1, \ldots, x_{k_1}^1, \ldots, x_1^n, \ldots, x_{k_n}^n\}$ and each $x_j^i \in \bar{X}$ occurs exactly once in $w_1 \cdots w_{k_0}$. Then $\sigma^{\mathcal{A}} \colon (\Delta^*)^{k_1} \times \cdots \times (\Delta^*)^{k_n} \to (\Delta^*)^{k_0}$ where

$$\sigma^{\mathcal{A}}((u_1^1, \ldots, u_{k_1}^1), \ldots, (u_1^n, \ldots, u_{k_n}^n)) = (w_1[x_j^i / u_j^i \mid x_j^i \in \bar{X}], \ldots, w_{k_0}[x_j^i / u_j^i \mid x_j^i \in \bar{X}])$$

As $\sigma^{\mathcal{A}}$ is uniquely determined by $w_1, \ldots, w_{k_0}$, we also denote $\sigma^{\mathcal{A}}$ by $\langle w_1, \ldots, w_{k_0} \rangle$. □

An LCFRS is obtained by combining an LCFRS algebra with a compatible RTG:

Definition 4.1.2. Let Δ be an alphabet and $(\Sigma, \mathrm{sort}_\Sigma)$ be an $\mathbb{N}^* \times \mathbb{N}$-sorted alphabet. A *(string) linear context-free rewriting system* (LCFRS) is an IRTG $(G, \mathcal{A})$ where

- $\mathcal{A}$ is a $((\Sigma, \mathrm{sort}_\Sigma), \Delta)$-LCFRS algebra,
- $G = (N, (\Sigma, \mathrm{rk}_\Sigma), S, R)$ is a $(\Sigma, \mathrm{sort}_\Sigma)$-compatible RTG, and
- $\mathrm{sort}_N(S) = 1$.

The *string language of* $(G, \mathcal{A})$, denoted by $\mathcal{L}(G, \mathcal{A})$, is $\{w \mid \xi \in L(G), [\![\xi]\!]_1^{\mathcal{A}} = (w)\}$. □

An important property of an LCFRS is its *fanout*: it is defined as the value

$$\max_{B \in N} \mathrm{sort}_N(B) \ ,$$

i.e., the maximal arity of tuples over which the LCFRS algebra operates. We also refer to the value $\mathrm{sort}_N(B)$ as *fanout of* B. The fanout influences the expressiveness and the parsing complexity of the LCFRS (see Chapter 6). An LCFRS with fanout one can be equivalently represented as a context-free grammar. The next example shows an LCFRS with fanout two.

Example 4.1.3. Let $\Sigma = \{\sigma, \gamma, \delta, \beta\}$ be an $\mathbb{N}^* \times \mathbb{N}$-sorted alphabet with $\mathrm{sort}_\Sigma(\sigma) = (2\ 2, 1)$, $\mathrm{sort}_\Sigma(\gamma) = (2, 2) = \mathrm{sort}_\Sigma(\delta)$, and $\mathrm{sort}_\Sigma(\beta) = (\varepsilon, 2)$. Let $\Delta = \{a, b, c, d\}$ be an alphabet. Let $\mathcal{A}$ be a (Σ, Δ)-LCFRS algebra where

$$\sigma^{\mathcal{A}} = \langle x_1^1 x_1^2 x_2^1 x_2^2 \rangle \ , \quad \gamma^{\mathcal{A}} = \langle a x_1^1, c x_2^1 \rangle \ , \quad \delta^{\mathcal{A}} = \langle b x_1^1, d x_2^1 \rangle \ , \quad \text{and} \quad \beta^{\mathcal{A}} = \langle \varepsilon, \varepsilon \rangle \ .$$

Let $G = (N, \Sigma, S, R)$ be an RTG with $N = \{S, A, B\}$ and

$$R = \{S \to \sigma(A, B), \quad A \to \gamma(A), \quad A \to \beta(), \quad B \to \delta(B), \quad B \to \beta()\} \ .$$

Observe that $\mathbb{G} = (G, \mathcal{A})$ is an LCFRS, $L(G) = \{\sigma(\gamma^m(\beta), \delta^n(\beta)) \mid m, n \in \mathbb{N}\}$, $[\![\gamma^m(\beta)]\!]_2^{\mathcal{A}} = (a^m, c^m)$ for each $m \in \mathbb{N}$, and $[\![\delta^n(\beta)]\!]_2^{\mathcal{A}} = (b^n, d^n)$ for each $n \in \mathbb{N}$. Thus, $\mathcal{L}(G, \mathcal{A}) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$. □

4.2 Definite clause programs

Definite clause programs (DCPs) are due to Deransart and Małuszyński (1985, 1989). They are a particular class of logic programs that can be used to describe multi-component tree languages, i.e., languages where multiple trees are generated synchronously. DCPs are related to tree-generating attribute grammars (Knuth 1968). In particular, they have a form of inherited and synthesized attributes, which allow to pass information from outside a particular subderivation to this subderivation. Nederhof and Vogler (2014) and Gebhardt, Nederhof, and Vogler (2017) employ so-called *simple* DCPs to model the generation of discontinuous and non-projective structures. The multi-component capability of DCP is used to generate, for a particular subsentence, a tuple of maximal connected trees that cover the subsentence. In the case of dependency parsing, the inherited attributes are utilized to inject missing child nodes into the trees. The property *simple* means that each argument (both inherited and synthesized ones) is used exactly once. A prominent problem when dealing with attribute grammars (as well as DCPs) is to check for and deal with cycles when evaluating attributes (Jazayeri, Ogden, and Rounds 1975; Räihä and Saarinen 1982; P.-C. Wu 2004): the inherited and synthesized attributes in general allow for mutual dependency of two attributes of another. In Gebhardt, Nederhof, and Vogler (2017) this issue is circumvented by encoding the dependencies into the grammar's nonterminals. Since the trees from which the grammar is induced are non-circular, the encoding suffices to guarantee that the grammar is non-circular. Apparently, the sDCPs considered by Gebhardt, Nederhof, and Vogler (2017) can even be viewed as *strongly non-circular* (i.e., attribute grammars that have no circular dependencies even if the attribute relationships of each nonterminal are merged to a single graph, see Kennedy and S. K. Warren 1976; Courcelle and Franchi-Zannettacci 1982).

In this thesis, we use an alternative definition for strongly non-circular simple definite clause programs based on initial algebra semantics to maintain consistency with the IRTG framework outlined before. In contrast, the semantics of sDCP used in Gebhardt, Nederhof, and Vogler (2017) involves a derivation relation based on rewriting nonterminals and their arguments. The sDCP algebra that we define uses tuples of contexts as domain. Operations are defined by the help of contexts $\xi_1, \ldots, \xi_l$ with second-order variables. The resulting operation $\sigma^{\mathcal{A}} = \langle \xi_1, \ldots, \xi_l \rangle$ yields the tuple $(\hat{\varphi}(\xi_1), \ldots, \hat{\varphi}(\xi_l))$ of hedges that is obtained by replacing each second-order variable with some context in one of $\sigma^{\mathcal{A}}$'s arguments. In this way, the behaviour of multiple components and inherited arguments is simulated.

We will not give a formal proof of the equivalence of simple DCP according to Deransart and Małuszyński (1985) to our definition. This would, on the one hand, require certain restrictions (strongly non-circular sDCP) and normal forms (deterministic dependencies between inherited and synthesized attributes). On the other hand, it would require us to recall the original formal definition and to give

involved technical proofs. As a weaker but for our purposes sufficient claim, we suppose that the sDCPs induced by Gebhardt, Nederhof, and Vogler (2017) can be equivalently represented with the new definition and that the new definition does not add in expressiveness to original sDCPs. We present the modified induction algorithms in the next chapter.

Before we can formally present sDCP, we define second-order substitution. Here, so-called second-order variables that occur at internal positions of a hedge ξ are substituted with hedge contexts specified by a function φ. If a second-order variable x is of rank k, then it is replaced by a hedge context with k variables $y_1, \ldots, y_k$. Note that x has k child hedges in ξ. Hence, we replace each variable y_i by the i-th child of x.

Definition 4.2.1 (Second-order substitution)**.** Let Δ be an alphabet and X be a ranked alphabet. Let $\varphi\colon X \to \bigcup_{n\in\mathbb{N}} \mathrm{U}^*_\Delta(Y_n)$ be such that $\varphi(x) \in \mathrm{U}^*_\Delta(Y_{\mathrm{rk}(x)})$ for each $x \in X$. The *second-order substitution according to* φ is the function $\hat{\varphi}\colon \mathrm{U}^*_\Delta(Y, X) \to \mathrm{U}^*_\Delta(Y)$ defined recursively as follows:

$$\hat{\varphi}(\xi) = \begin{cases} \hat{\varphi}(\xi_1)\cdots\hat{\varphi}(\xi_n) & \text{if } \xi = \xi_1\cdots\xi_n \text{ for } n \neq 1,\ \xi_1,\ldots,\xi_n \in \mathrm{U}_\Delta(Y,X) \\ \xi & \text{if } \xi \in Y \\ \delta(\hat{\varphi}(\xi')) & \text{if } \xi = \delta(\xi') \text{ for } \delta \in \Delta,\ \xi' \in \mathrm{U}^*_\Delta(Y,X) \\ \varphi(x)[y_i/\hat{\varphi}(\xi_i) \mid i \in [k]] & \text{if } \xi = x(\xi_1,\ldots,\xi_k) \text{ for } k \in \mathbb{N}, x \in X_k, \\ & \qquad \xi_1,\ldots,\xi_k \in \mathrm{U}^*_\Delta(Y,X) \end{cases}$$

□

Example 4.2.2. Let $\Delta = \{a, b, c\}$ be an alphabet and let $X = \{x^1_1, x^1_2, x^1_3\}$ be a ranked alphabet with $\mathrm{rk}(x^1_1) = 2 = \mathrm{rk}(x^1_2)$ and $\mathrm{rk}(x^1_3) = 0$. Figure 4.1 shows a hedge ξ in $\mathrm{C}^*_\Delta(Y_2, X)$, a function $\varphi\colon X \to \mathrm{U}^*_\Delta(Y)$, and the evaluation of $\hat{\varphi}(\xi)$. □

Observation 4.2.3. Let $\xi \in \mathrm{C}^*_\Delta(Y, X)$ be such that each $x \in X$ occurs at most once in ξ. Let φ be such that $\varphi(x) \in \mathrm{C}^*_\Delta(Y_{\mathrm{rk}(x)})$ for each $x \in X$. It holds that $\hat{\varphi}(\xi) \in \mathrm{C}^*_\Delta(Y)$. □

By using the notion of second-order substitution and the preceding observation, we can define the sDCP algebra. It operates over tuples of hedge contexts of different arities. Hence, the sorts specify the arity of the tuple and for each tuple position the number of variables in the context. In order to specify the simple, i.e., linear and non-deleting, operations, we use tuples of prototypic hedge contexts with second-order variables.

Definition 4.2.4. Let Σ be an $(\mathbb{N}^*)^* \times \mathbb{N}^*$-sorted alphabet and Δ be an alphabet. A (Σ, Δ)-*sDCP algebra* is a Σ-algebra $\mathcal{A}$ where $\mathcal{A}_{s^0_1\cdots s^0_{l_0}} = (\mathrm{C}^*_\Delta(Y_{s^0_1}) \times \cdots \times \mathrm{C}^*_\Delta(Y_{s^0_{l_0}}))$ and operations are as follows:

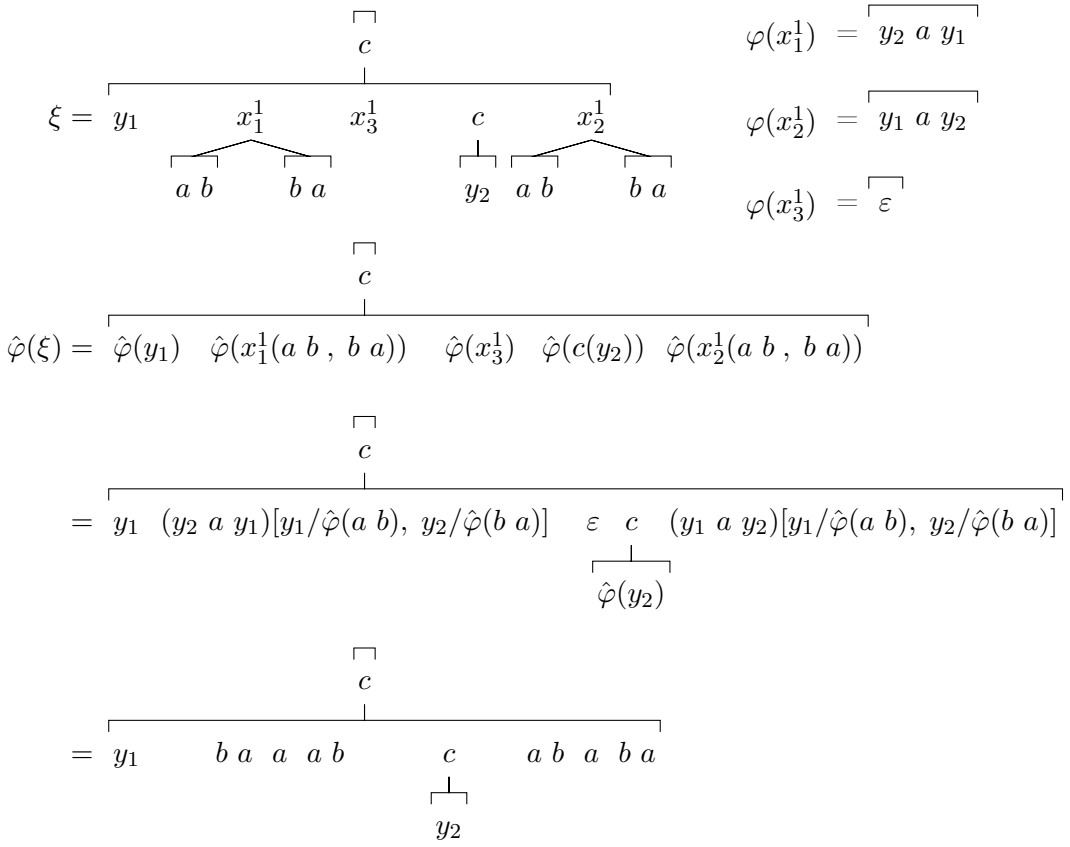


Figure 4.1: Second order substitution: A hedge ξ, an instance of the function φ, and the evaluation of $\hat{\varphi}(\xi)$.

For each $k \in \mathbb{N}$, $s_1^1, \ldots, s_{l_1}^1, \ldots, s_1^k, \ldots, s_{l_k}^k, s_1^0, \ldots, s_{l_0}^0 \in \mathbb{N}$, and

$$\sigma \in \Sigma_{((s_1^1 \cdots s_{l_1}^1) \cdots (s_1^k \cdots s_{l_k}^k), (s_1^0 \cdots s_{l_0}^0))}$$

there are

$$\xi_1 \in \mathrm{C}^*_\Delta(Y_{s_1^0}, X), \ \ldots, \ \xi_{l_0} \in \mathrm{C}^*_\Delta(Y_{s_{l_0}^0}, X)$$

where $X = \{x_1^1, \ldots, x_{l_1}^1, \ldots, x_1^k, \ldots, x_{l_k}^k\}$ and, for each $i \in [k]$ and $j \in [l_k]$, we have that

- $\mathrm{rk}(x_j^i) = s_j^i$, and
- x_j^i occurs exactly once in $\xi_1 \cdots \xi_{l_0}$.

Then, for every $\zeta_1^1 \in \mathrm{C}^*_\Delta(Y_{s_1^1}), \ldots, \zeta_{l_1}^1 \in \mathrm{C}^*_\Delta(Y_{s_{l_1}^1}), \ldots, \zeta_1^k \in \mathrm{C}^*_\Delta(Y_{s_1^k}), \ldots, \zeta_{l_k}^k \in \mathrm{C}^*_\Delta(Y_{s_{l_k}^k})$, we define

$$\sigma^{\mathcal{A}}((\zeta_1^1, \ldots, \zeta_{l_1}^1), \ldots, (\zeta_1^k, \ldots, \zeta_{l_k}^k)) = (\hat{\varphi}(\xi_1), \ldots, \hat{\varphi}(\xi_{l_0})) \ ,$$

where we set $\varphi(x_j^i) = \zeta_j^i$ for every $i \in [k]$ and $j \in [l_i]$. We denote $\sigma^{\mathcal{A}}$ by $\langle \xi_1, \ldots, \xi_{l_0} \rangle$. □

Finally, an sDCP algebra combined with a compatible RTG constitutes an sDCP.

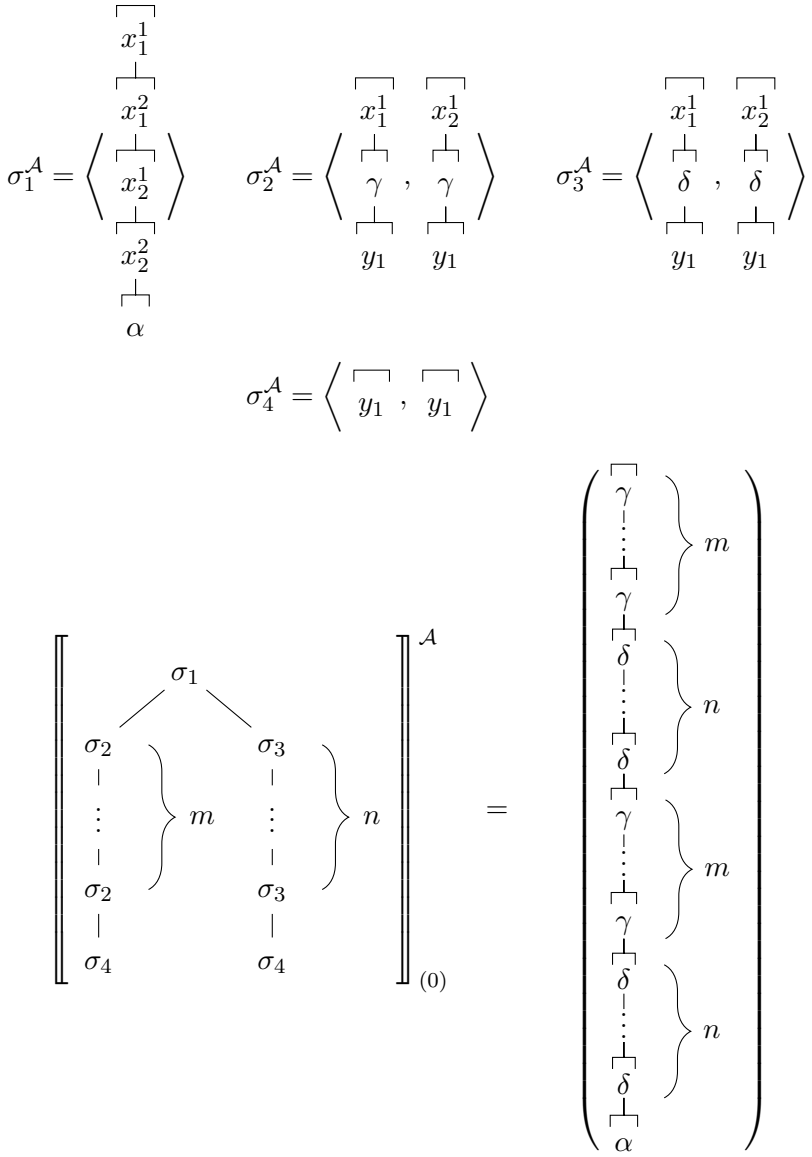
Definition 4.2.5. Let Δ be an alphabet and $(\Sigma, \mathrm{sort}_\Sigma)$ be an $(\mathbb{N}^*)^* \times \mathbb{N}^*$-sorted alphabet. A *(strongly non-circular) simple definite clause program* (sDCP) is an IRTG $(G, \mathcal{A})$ where

- $\mathcal{A}$ is a $((\Sigma, \mathrm{sort}_\Sigma), \Delta)$-sDCP algebra,
- $G = (N, (\Sigma, \mathrm{rk}_\Sigma), S, R)$ is a $(\Sigma, \mathrm{sort}_\Sigma)$-compatible RTG, and
- $\mathrm{sort}_N(S) = (0)$.

The *hedge language of* $(G, \mathcal{A})$, denoted by $\mathcal{L}(G, \mathcal{A})$, is $\{\xi \mid t \in L(G), [\![t]\!]_0^{\mathcal{A}} = (\xi)\}$. □

We give a small example of the expressive power of sDCP where a language of monadic trees is generated. Note that the corresponding *path language*, i.e., the words that are obtained if we read symbols on paths from the root to some leaf in a tree, can also be specified by an LCFRS similar to the one in Example 4.1.3.

Example 4.2.6. Let $\Sigma = \{\sigma_1, \ldots, \sigma_4\}$ be an $(\mathbb{N}^*)^* \times \mathbb{N}^*$-sorted alphabet (with sorts indicated below), $\Delta = \{\gamma, \delta, \alpha\}$ be an alphabet, and $\mathcal{A}$ be a (Σ, Δ)-sDCP algebra



$$\begin{aligned}
\sigma_1^{\mathcal{A}}(\sigma_2^{\mathcal{A}}(\sigma_4^{\mathcal{A}}), \sigma_3^{\mathcal{A}}(\sigma_4^{\mathcal{A}})) &= \sigma_1^{\mathcal{A}}(\sigma_2^{\mathcal{A}}(\ (y_1, y_1)\ ), \sigma_3^{\mathcal{A}}(\ (y_1, y_1)\ )) \\
&= \sigma_1^{\mathcal{A}}(\ (\gamma(y_1), \gamma(y_1))\ ,\ (\delta(y_1), \delta(y_1))\ ) \\
&= \gamma(\delta(\gamma(\delta(\alpha))))
\end{aligned}$$

Figure 4.2: Operations of an sDCP algebra $\mathcal{A}$, the interpretation of a tree, and stepwise evaluation of operations for the case $m = 1$ and $n = 1$.

with operations specified in Figure 4.2. Let G be an RTG given by the following rules:

$$
\begin{array}{lllll}
G: & S \to \sigma_1(A, B) & \mathrm{sort}(\sigma_1) = ((1\ 1)(1\ 1), (0)) & & \\
 & A \to \sigma_2(A) + \sigma_4() & \mathrm{sort}(\sigma_2) = ((1\ 1), (1\ 1)) & \mathrm{sort}(\sigma_4) = (\varepsilon, (1\ 1)) & \\
 & B \to \sigma_3(B) + \sigma_4() & \mathrm{sort}(\sigma_3) = ((1\ 1), (1\ 1)) & &
\end{array}
$$

Then $(G, \mathcal{A})$ is an sDCP. The evaluation of the trees from $L(G)$ in $\mathcal{A}$ is depicted in Figure 4.2. □

4.3 An alignment algebra for LCFRS and sDCP

Our goal is to couple an LCFRS and an sDCP algebra in a single IRTG. However, if we just have these two algebras together with an RTG, then the result is not yet a hybrid grammar but a so-called *synchronous grammar* (Shieber and Schabes 1990; Satta and Peserico 2005). The main difference between hybrid grammars and synchronous grammars is that the latter only synchronize derivational nonterminal symbols whereas in the former also the terminal symbols are synchronized. The IRTG framework was designed to capture synchronous grammars by pulling out the shared structure of the synchronized rules into an RTG and representing the different semantics by means of algebras. However, if alignments between terminal symbols generated by the different algebras shall be modelled, then something needs to be added. In principle, there are two options:

1. All involved algebras $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and their domains are merged such that a new algebra $\mathcal{A}$ with domain $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k \times \mathcal{S}$ is obtained. Here, $\mathcal{S}$ is a new dimension that will contain the alignments between objects from the other dimensions. Each operation $\sigma^{\mathcal{A}}$ can access each dimension of $\mathcal{A}$ and update alignments appropriately.

   We have explored this path in Drewes, Gebhardt, and Vogler (2016). Specifically, we represent LCFRS and sDCP as graph grammars and enforce that the sets of nodes of the graphs generated in either algebra are disjoint. Alignments are encoded by additional edges between nodes of both graphs. An operation of a graph grammar embeds a list of argument graphs with pairwise disjoint sets of nodes into a larger graph. Without loss of generality, one assumes that all operations are on pairwise disjoint graphs because nodes can always be renamed. This reduces the technical load in the specification but comes at a computational cost in the implementation.

   I suppose that by merging the algebras a part of the elegance of initial algebra semantics gets lost. Moreover, for practical applications such as solving the IRTG parsing problem one either needs to blow up the IRTG by copies of

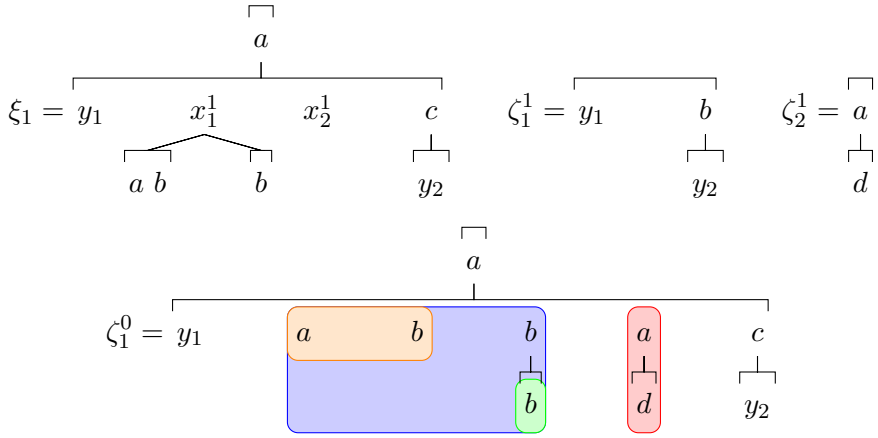


Figure 4.3: Four hedges.

the original algebras (as in Gebhardt 2018) or make ad-hoc projections (as in Gebhardt, Nederhof, and Vogler 2017, Algorithm 5.1).

2. Alternatively, one just adds a new *alignment algebra* $\mathcal{A}_{\text{align}}$. The domain of this algebra contains only the minimal required information about the shape of the objects in $\mathcal{A}_1, \ldots, \mathcal{A}_k$ as well as which positions in these objects are synchronized. This way, the original (non-synchronized) algebras remain unaltered.

We explore the second approach in this thesis. While representing the shape of the domain of an LCFRS algebra is very simple (it suffices to remember the length λ_i of the i-th string in the tuple (for each $i \in [k]$)), the same turns out quite technical for sDCP: Each operation takes hedge contexts (organized in tuples) as inputs and builds new hedge contexts by substituting each second-order variable that occurs in some skeleton hedge. For instance, consider the hedges ξ_1, ζ_1^1, ζ_2^1, and ζ_1^0 depicted in Figure 4.3. If we apply the operation $\langle \xi_1 \rangle$ to (ζ_1^1, ζ_2^1), then in ξ_1, x_1^1 is substituted by ζ_1^1, x_2^1 is substituted by ζ_2^1, and we obtain (ζ_1^0). Thus, ζ_1^1 and ζ_2^1 are part of the resulting hedge ζ_1^0 (see the blue and the red box, respectively) but their positions get shifted for two reasons:

- x_1^1 and x_2^1 are not at position ε in ξ_1, i.e., the first position at root level. For instance, in order to obtain the positions in ζ_1^0 which correspond to ζ_1^1, we have to prepend some offset to the positions in ζ_1^1. This offset is the position of x_1^1 in ξ_1, i.e., $\downarrow\rightarrow$. As x_2^1 occurs to the right of x_1^1 in ξ_1, the offset for positions in ζ_2^1 additionally needs to account for the length of ζ_1^1.

- The (first-order) variables y_1 and y_2 in ζ_1^1 are replaced by the children of x_1^1 (see the orange and green box in ζ_1^0, respectively). In our example the first child

p	$\mathrm{length}^{\xi_1}_{\theta,\mu}(p)$	$\mathbf{t}\mathrm{off}^{\xi_1}_{\theta,\mu,\varrho}(p)$
$\diamond$	1	ε
$\downarrow\diamond$	$\theta(y_1)$	$\downarrow$
$\downarrow\rightarrow\diamond$	$\mu^1_1(\{y_1 \mapsto 2, y_2 \mapsto 1\})$	$\downarrow\rightarrow^{\theta(y_1)}$
$\downarrow\rightarrow_1\searrow\diamond$	1	$\downarrow\rightarrow^{\theta(y_1)} \cdot \varrho^1_1(\{y_1 \mapsto 2, y_2 \mapsto 1\})(1) \cdot \varepsilon$
$\downarrow\rightarrow_1\searrow\rightarrow\diamond$	1	$\downarrow\rightarrow^{\theta(y_1)} \cdot \varrho^1_1(\{y_1 \mapsto 2, y_2 \mapsto 1\})(1) \cdot \rightarrow$
$\downarrow\rightarrow_2\searrow\diamond$	1	$\downarrow\rightarrow^{\theta(y_1)} \cdot \varrho^1_1(\{y_1 \mapsto 2, y_2 \mapsto 1\})(2) \cdot \varepsilon$
$\downarrow\rightarrow^2\diamond$	$\mu^1_2(\emptyset)$	$\downarrow\rightarrow^{\theta(y_1)+\mu^1_1(\{y_1\mapsto 2, y_2\mapsto 1\})}$
$\downarrow\rightarrow^3\diamond$	1	$\downarrow\rightarrow^{\theta(y_1)+\mu^1_1(\{y_1\mapsto 2, y_2\mapsto 1\})+\mu^1_2(\emptyset)}$
$\downarrow\rightarrow^3\downarrow\diamond$	$\theta(y_2)$	$\downarrow\rightarrow^{\theta(y_1)+\mu^1_1(\{y_1\mapsto 2, y_2\mapsto 1\})+\mu^1_2(\emptyset)}\downarrow$

Table 4.4: Future lengths and offsets for the hedge ξ_1 in Figure 4.3.

has length 2. Hence, the node labeled b in ζ^1_1, which occurs to the right of y_1, gets an additional shift of $\rightarrow^2$.

However, notice that ζ^0_1 is a context itself, which we will later want to substitute for a second-order variable. Thus, we can only calculate these shifts modulo the size of the hedges that we substitute for y_1 and y_2 in ζ^0_1. We solve this issue by assuming a function θ_1 that abstracts away from the lengths of these hedges that we do not yet know.

Turning again to the general case, we model the shape of a tuple of contexts $(\zeta^0_1, \ldots, \zeta^0_l)$ by two families of higher-order functions. For each $i \in [l]$, μ_i computes the length of ζ^0_i if each variable y_o in ζ^0_i is substituted by a hedge of length $\theta_i(y_o)$. Likewise, if we substitute each variable y_o of ζ^0_i by a hedge ξ'_o of length $\theta_i(y_o)$, then ϱ_i conveys us the offset of ξ'_o in $\zeta^0_i[\xi'_o/o \in [s^0_o]]$. Finally, we represent the alignment that links some position in a tuple of strings to ζ^0_i by a partial higher-order function α_i. α_i maps a pair consisting of a tuple index and a string position to some position in ζ^0_i, again requiring also θ_i as input.

In order to define μ, ϱ, and α, we first introduce auxiliary functions that recursively compute the lengths and offsets for hedge positions. The first function $\mathrm{length}^{\xi}(p)$ computes to how many symbols a position p in ξ will be expanded due to substitution of first-order and second-order variables.

Definition 4.3.1. Let $\xi \in \mathrm{U}^*_{\Delta}(Y, X)$, $p \in \mathrm{pos}(\xi)$, $\theta\colon Y \to \mathbb{N}$, and, for each $x^q_r \in X$, let $\mu^q_r\colon (Y_{\mathrm{rk}_X(x^q_r)} \to \mathbb{N}) \to \mathbb{N}$. We define the *future length of* ξ *at* p *given* θ *and* μ as follows:

$$\mathrm{length}^{\xi}_{\theta,\mu}(p) = \begin{cases} 1 & \text{if } \xi(p) \in \Delta \\ \theta(y_j) & \text{if } \xi(p) = y_j \text{ in } Y \\ \mu^q_r(\psi^{\xi}_{\theta,\mu}(p)) & \text{if } \xi(p) = x^q_r \text{ in } X \end{cases} ,$$

where, if $\xi(p) = x^q_r$ in X with $p = p'\diamond$, we let $\psi^{\xi}_{\theta,\mu}(p)\colon Y_{\mathrm{rk}_X(x^q_r)} \to \mathbb{N}$ be such that, for each $j \in [\mathrm{rk}_X(x^q_r)]$, we have

$$(\psi^{\xi}_{\theta,\mu}(p))(y_j) = \sum_{j' \in \mathbb{N}\colon (p' {}_{j}\!\searrow \to^{j'} \diamond) \in \mathrm{pos}(\xi)} \mathrm{length}^{\xi}_{\theta,\mu}(p' {}_{j}\!\searrow \to^{j'} \diamond) \ .$$

For brevity, we also define the notion

$$\mathrm{length}^{\xi}_{\theta,\mu}(< p) = \sum_{\substack{p'\colon p'\diamond \in \mathrm{pos}(\xi) \\ \exists j>0\colon p = p' \to^j \diamond}} \mathrm{length}^{\xi}_{\theta,\mu}(p'\diamond) \ ,$$

where we accumulate the length of all positions to the left of p. □

The values of $\mathrm{length}^{\xi_1}_{\theta,\mu}(p)$ for the hedge ξ_1 from Figure 4.3 and different positions $p \in \mathrm{pos}(\xi_1)$ are given in Table 4.4.

Remark 4.3.2. $\mathrm{length}^{\xi}_{\theta,\mu}(p)$ (and $\psi^{\xi}_{\theta,\mu}(p)$) can be evaluated for all p in linear time in the size of ξ via a bottom-up traversal over ξ. □

Next, we define a function $\mathbf{t}\mathrm{off}$, that reflects the offset for each position in the tree due to substitution.

Definition 4.3.3. Let $\xi \in \mathrm{U}^*_{\Delta}(Y, X)$, $p \in \mathrm{pos}(\xi)$, $\theta\colon Y \to \mathbb{N}$, and, for each $x^q_r \in X$, let $\mu^q_r\colon (Y_{\mathrm{rk}_X(x^q_r)} \to \mathbb{N}) \to \mathbb{N}$ and $\varrho^q_r\colon (Y_{\mathrm{rk}_X(x^q_r)} \to \mathbb{N}) \to \{\to, \downarrow\}^*$. We define $\mathbf{t}\mathrm{off}^{\xi}_{\theta,\mu,\varrho}\colon \mathrm{pos}(\xi) \to \{\to, \downarrow\}^*$ such that

$$\mathbf{t}\mathrm{off}^{\xi}_{\theta,\mu,\varrho}(p) = \begin{cases} \to^{(\mathrm{length}^{\xi}_{\theta,\mu}(<p))} & \text{if } p \in \{\to\}^* \cdot \{\diamond\} \\ \mathbf{t}\mathrm{off}^{\xi}_{\theta,\mu,\varrho}(p'\diamond) \cdot \downarrow \cdot \to^{\mathrm{length}^{\xi}_{\theta,\mu}(<p)} & \text{if } p = p' \downarrow \to^j \diamond \\ & \text{with } p'\diamond \in \mathrm{pos}(\xi) \wedge j \in \mathbb{N} \\ \mathbf{t}\mathrm{off}^{\xi}_{\theta,\mu,\varrho}(p'\diamond) \cdot \varrho^q_r(\psi^{\xi}_{\theta,\mu}(p'\diamond))(j_1) & \text{if } p = p' {}_{j_1}\!\searrow \to^{j_2} \diamond \\ \quad \cdot \to^{\mathrm{length}^{\xi}_{\theta,\mu}(<p)} & \wedge\, \xi(p'\diamond) = x^q_r \text{ in } X \\ & \text{with } p'\diamond \in \mathrm{pos}(\xi) \wedge j_1, j_2 \in \mathbb{N} \ . \end{cases}$$

□

Again, Table 4.4 shows $\mathbf{t}\mathrm{off}^{\xi_1}$ for the positions in ξ_1 given in Figure 4.3.

Remark 4.3.4. $\mathbf{t}\mathrm{off}^{\xi}_{\theta,\mu,\varrho}$ can be evaluated for each $p \in \mathrm{pos}(\xi)$ in linear time in the size of ξ via a top-down traversal over ξ given that $\mathrm{length}^{\xi}_{\theta,\mu}(p)$ has been computed before for each $p \in \mathrm{pos}(\xi)$. □

Having prepared these auxiliary definitions, we turn to defining the alignment algebra.

Definition 4.3.5. Let $\mathcal{A}_1$ be a $((\Sigma, \mathrm{sort}_{\mathrm{LCFRS}}), \Gamma)$-LCFRS algebra and let $\mathcal{A}_2$ be a $((\Sigma, \mathrm{sort}_{\mathrm{sDCP}}), \Delta)$-sDCP algebra such that $\mathrm{sort}_{\mathrm{LCFRS}}$ and $\mathrm{sort}_{\mathrm{sDCP}}$ are compatible.

An $(\mathcal{A}_1, \mathcal{A}_2)$-*alignment algebra* is a $(\Sigma, \mathrm{sort}_{\mathrm{align}})$-algebra $(\mathcal{A}_3, \cdot^{\mathcal{A}_3})$ where $\mathrm{sort}_{\mathrm{align}} = \mathrm{sort}_{\mathrm{LCFRS}} \times \mathrm{sort}_{\mathrm{sDCP}}$ and $\mathcal{A}_3$ and $\cdot^{\mathcal{A}_3}$ are as follows.

Let $w = (k, s_1 \cdots s_l) \in \mathbb{N} \times \mathbb{N}^*$. Then

$$\begin{aligned}(\mathcal{A}_3)_w = \{(\lambda, \mu, \varrho, \alpha) \mid\ & \lambda = (\lambda_j \in \mathbb{N} \mid j \in [k]),\\ & \mu = (\mu_j \colon (Y_{s_j} \to \mathbb{N}) \to \mathbb{N} \mid j \in [l]),\\ & \varrho = (\varrho_j \colon (Y_{s_j} \to \mathbb{N}) \to [s_j] \to \{\rightarrow, \downarrow\}^* \mid j \in [l]),\\ & \alpha = (\alpha_j \colon (Y_{s_j} \to \mathbb{N}) \to ([k] \times \mathbb{N}) \dashrightarrow \{\rightarrow, \downarrow, \diamond\}^* \mid j \in [l]) \quad \} \ .\end{aligned}$$

Furthermore, for each $\sigma \in \Sigma$ where

- $\mathrm{sort}_{\mathrm{align}}(\sigma) = ((k_1, s^1_1 \cdots s^1_{l_1}) \ldots (k_n, (s^n_1 \cdots s^n_{l_n})), (k_0, (s^0_1 \cdots s^0_{l_0})))$,
- $\sigma^{\mathcal{A}_1} = \langle w_1, \ldots, w_{k_0} \rangle$, and
- $\sigma^{\mathcal{A}_2} = \langle \xi_1, \ldots, \xi_{l_0} \rangle$

there is an injective function

$$\alpha \colon \{(i,j) \mid i \in [k_0], j \in [|w_i|] \colon w_i[j] \in \Gamma\} \to \{(i,p) \mid i \in [l_0], p \in \mathrm{pos}_{\Delta}(\xi_i)\} \ .$$

We define the function

$$\sigma_3^{\mathcal{A}}((\lambda^1, \mu^1, \varrho^1, \alpha^1), \ldots, (\lambda^n, \mu^n, \varrho^n, \alpha^n)) = (\lambda^0, \mu^0, \varrho^0, \alpha^0)$$

in multiple steps. Specifically, λ^0 is such that,

- for each $i \in [k]$, $\lambda^0_i = \mathrm{off}^i_{\mathrm{s}}(|w_i|)$ where
- $\mathrm{off}^i_{\mathrm{s}} \colon [|w_i|]_0 \to \mathbb{N}$ satisfies

$$\mathrm{off}^i_{\mathrm{s}}(j) = \sum_{j'=1}^{j} \begin{cases} 1 & \text{if } w_i[j'] \in \Gamma \\ \lambda^q_r & \text{if } w_i[j'] = x^q_r \end{cases} \ .$$

For each $i \in [l_0]$, let $\theta_i \colon Y_{s^0_i} \to \mathbb{N}$. We set

- $\mu_i^0(\theta_i) = \sum_{j\in\mathbb{N}:\ \to^j\diamond\in\mathrm{pos}(\xi_i)} \mathrm{length}^{\xi_i}_{\theta_i,\mu}(\to^j\diamond)$.
- For each $j \in [s_i^0]$, we set $\varrho_i^0(\theta_i)(j) = \mathbf{t}\mathrm{off}^{\xi_i}_{\theta_i,\mu,\varrho}(p)$ where $p \in \mathrm{pos}(\xi_i)$ is such that $\xi_i(p) = y_j$.
- For each $(i,j) \in \mathrm{dom}(\alpha)$ with $\alpha(i,j) = (m,p)$, we set

$$\alpha_m^0(\theta_m)(i, \mathrm{off}^i_{\mathrm{s}}(j)) = \mathbf{t}\mathrm{off}^{\xi_m}_{\theta_m,\mu,\varrho}(p)\ \diamond\ .$$

- For each $q \in [n]$, $m \in [l_q]$, and $(i,j) \in \mathrm{dom}(\alpha^q_m(\tau))$ with[1]
  - $i' \in [k_0]$ and $j' \in [|w_{i'}|]$ such that $x_i^q = w_{i'}[j']$ and
  - $m' \in [l_0]$ and $p \in \mathrm{pos}(\xi_{m'})$ such that $\xi_{m'}(p) = x_m^q$

  we set

$$\alpha^0_{m'}(\theta_{m'})(i', \mathrm{off}^{i'}_{\mathrm{s}}(j'-1)+j) = \mathbf{t}\mathrm{off}^{\xi_{m'}}_{\theta_{m'},\mu,\varrho}(p) \cdot \alpha^q_m(\psi^{\xi_{m'}}_{\theta_{m'},\mu}(p))(i,j)\ .$$

We denote $\sigma^{\mathcal{A}_3}$ by $\langle w_1,\dots,w_{k_0};\xi_1,\dots,\xi_{l_0};\alpha\rangle$. □

Example 4.3.6. Consider the alphabet $\Sigma = \{\gamma,\beta\}$ and the functions $\mathrm{sort}_{\mathrm{LCFRS}}$ and $\mathrm{sort}_{\mathrm{sDCP}}$ with:

$$\begin{array}{ll} \mathrm{sort}_{\mathrm{LCFRS}}\colon \Sigma \to \mathbb{N}^*\times\mathbb{N} & \mathrm{sort}_{\mathrm{sDCP}}\colon \Sigma \to (\mathbb{N}^*)^*\times\mathbb{N}^* \\ \mathrm{sort}_{\mathrm{LCFRS}}(\gamma) = (2,1) & \mathrm{sort}_{\mathrm{sDCP}}(\gamma) = ((2\ 0),(2)) \\ \mathrm{sort}_{\mathrm{LCFRS}}(\beta) = (\varepsilon,2) & \mathrm{sort}_{\mathrm{sDCP}}(\beta) = (\varepsilon,(2\ 0))\ . \end{array}$$

Note that $\mathrm{sort}_{\mathrm{LCFRS}}$ and $\mathrm{sort}_{\mathrm{sDCP}}$ are compatible. Let $\Delta = \{a,b,c,d\}$ be an alphabet. Let $\mathcal{A}_1$ be a $((\Sigma,\mathrm{sort}_{\mathrm{LCFRS}}),\Delta)$-LCFRS algebra and $\mathcal{A}_2$ be a $((\Sigma,\mathrm{sort}_{\mathrm{sDCP}}),\Delta)$-sDCP algebra where

$$\begin{array}{ll} \gamma^{\mathcal{A}_1} = \langle x_1^1 c x_2^1\rangle & \gamma^{\mathcal{A}_2} = \langle \xi_1\rangle \\ \beta^{\mathcal{A}_1} = \langle b\ a, d\rangle & \beta^{\mathcal{A}_2} = \langle \zeta_1^1, \zeta_2^1\rangle \end{array}$$

with ξ_1, ζ_1^1, and ζ_2^1 as in Figure 4.3.
We define the $(\mathcal{A}_1,\mathcal{A}_2)$-alignment algebra $\mathcal{A}_3$ where

$$\gamma^{\mathcal{A}_3} = \langle\ x_1^1 c x_2^1\ ;\ \xi_1\ ;\quad \{(1,2)\mapsto(1,{\downarrow}{\to}^3\diamond)\}\ \rangle$$
$$\beta^{\mathcal{A}_3} = \langle\ b\ a\ ,\ d\ ;\ \zeta_1^1\ ,\ \zeta_2^1\ ;\ \{(1,1)\mapsto(1,{\to}\diamond),(1,2)\mapsto(2,\diamond),(2,1)\mapsto(2,{\downarrow}\diamond)\}\ \rangle\ .$$

[1] τ is an arbitrary function of type $Y_{s_m^q}\to\mathbb{N}$.

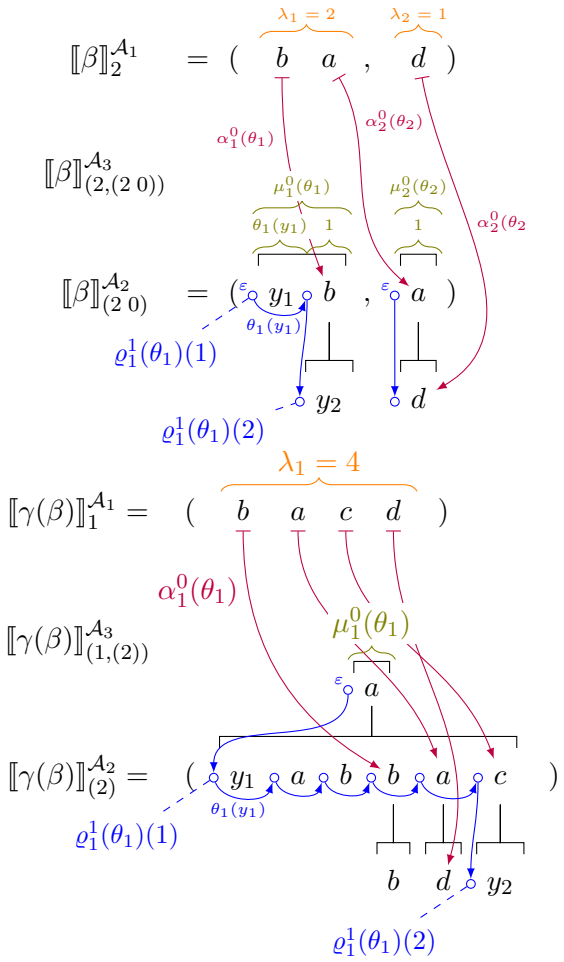


Figure 4.5: An alignment algebra at work.

Consider the tree $\gamma(\beta) \in (\mathrm{T}_\Sigma)_{(1,(2))}$. We first evaluate $[\![\beta]\!]^{\mathcal{A}_3}_{(2,(2,0))} = (\lambda^0, \mu^0, \gamma^0, \alpha^0)$ where

$$\begin{aligned}
\lambda^0_1 &= 2 & \lambda^0_2 &= 1\\
\mu^0_1(\theta_1) &= \theta_1(y_1) + 1 & \mu^0_2(\theta_2) &= 1\\
\varrho^0_1(\theta_1)(1) &= \varepsilon & \varrho^0_1(\theta_1)(2) &= {\rightarrow}^{\theta_1(y_1)}\\
\varrho^0_2(\theta_2) &= \emptyset\\
\alpha^0_1(\theta_1)(1,1) &= \mathbf{toff}^{\zeta^1_1}_{\theta_1,\mu,\varrho}(\rightarrow \diamond) \cdot \diamond = {\rightarrow}^{\theta_1(y_1)}\diamond\\
\alpha^0_2(\theta_2)(1,2) &= \mathbf{toff}^{\zeta^1_2}_{\theta_2,\mu,\varrho}(\diamond) \cdot \diamond = \diamond\\
\alpha^0_2(\theta_2)(2,1) &= \mathbf{toff}^{\zeta^1_2}_{\theta_2,\mu,\varrho}(\downarrow \diamond) \cdot \diamond = \downarrow\diamond\ .
\end{aligned}$$

These values are illustrated in Figure 4.5 on the left. In consequence, we obtain $(\lambda^0, \mu^0, \gamma^0, \alpha^0) = [\![\gamma(\beta)]\!]^{\mathcal{A}_3}_{(1,(2))} = \gamma^{\mathcal{A}_3}((\lambda^1, \mu^1, \gamma^1, \alpha^1))$ with $(\lambda^1, \mu^1, \gamma^1, \alpha^1) = [\![\beta]\!]^{\mathcal{A}_3}_{(2,(2,0))}$ as follows:

$$\begin{aligned}
\lambda^0_1 &= 4\\
\mu^0_1(\theta_1) &= 1\\
\varrho^0_1(\theta_1)(1) &= \mathbf{toff}^{\xi_1}_{\theta_1,\mu,\varrho}(\downarrow \diamond) = \downarrow\\
\varrho^0_1(\theta_1)(2) &= \mathbf{toff}^{\xi_1}_{\theta_1,\mu,\varrho}(\downarrow{\rightarrow}^3\downarrow \diamond)\\
&= \downarrow{\rightarrow}^{(\theta_1(y_1)+\mu^1_1(\{y_1\mapsto 2, y_2\mapsto 1\})+\mu^1_2(\emptyset))}\downarrow\\
&= \downarrow{\rightarrow}^{(\theta_1(y_1)+(2+1)+1)}\downarrow\\
\alpha^0_1(\theta_1)(1,2+1) &= \mathbf{toff}^{\xi_1}_{\theta_1,\mu,\varrho}(\downarrow{\rightarrow}^3 \diamond) \cdot \diamond\\
&= \downarrow{\rightarrow}^{(\theta_1(y_1)+\mu^1_1(\{y_1\mapsto 2, y_2\mapsto 1\})+\mu^1_2(\emptyset))} \cdot \diamond\\
&= \downarrow{\rightarrow}^{(\theta_1(y_1)+(2+1))+1)}\diamond\\
\alpha^0_1(\theta_1)(1,0+1) &= \mathbf{toff}^{\xi_1}_{\theta_1,\mu,\varrho}(\downarrow{\rightarrow} \diamond) \cdot \alpha^1_1(\{y_1 \mapsto 2, y_1 \mapsto 1\})(1,1)\\
&= \downarrow{\rightarrow}^{\theta_1(y_1)} \cdot {\rightarrow}^{\{y_1\mapsto 2, y_1\mapsto 1\}(y_1)}\diamond\\
&= \downarrow{\rightarrow}^{\theta_1(y_1)+2}\diamond\\
\alpha^0_1(\theta_1)(1,0+2) &= \mathbf{toff}^{\xi_1}_{\theta_1,\mu,\varrho}(\downarrow{\rightarrow}^2 \diamond) \cdot \alpha^1_2(\emptyset)(1,2)\\
&= \downarrow{\rightarrow}^{\theta_1(y_1)+\mu^1_1(\{y_1\mapsto 2, y_2\mapsto 1\})} \cdot \diamond\\
&= \downarrow{\rightarrow}^{\theta_1(y_1)+(2+1)} \cdot \diamond\\
\alpha^0_1(\theta_1)(1,3+1) &= \mathbf{toff}^{\xi_1}_{\theta_1,\mu,\varrho}(\downarrow{\rightarrow}^2 \diamond) \cdot \alpha^1_2(\emptyset)(2,1)\\
&= \downarrow{\rightarrow}^{\theta_1(y_1)+(2+1)} \cdot \downarrow\diamond
\end{aligned}$$

These values are illustrated in Figure 4.5 on the right. □

We conclude this section with two theorems and a corollary, which state that the alignment algebra is well-defined. The first theorem informs us that the shape tracking of the alignment algebra is correct. Precisely, we have that

(a) each λ_i has as value the length of the corresponding string generated by the LCFRS algebra,

(b) each μ_i has as value the length (modulo variable substitutions) of the corresponding hedge generated by the sDCP algebra, and that

(c) each ϱ_i can convey us the span position to left of a particular variable within a certain hedge generated by the sDCP algebra.

A proof is given in Appendix A.2 on pp. 205.

Theorem 4.3.7. Let $t \in (\mathrm{T}_\Sigma)_{(k_0, s_1^0 \cdots s_{l_0}^0)}$. If

- $(u_1^0, \ldots, u_{k_0}^0) = [\![t]\!]_{k_0}^{\mathcal{A}_1}$,
- $(\zeta_1^0, \ldots, \zeta_{l_0}^0) = [\![t]\!]_{(s_1^0 \ldots s_{l_0}^0)}^{\mathcal{A}_2}$, and
- $(\lambda^0, \mu^0, \varrho^0, \alpha^0) = [\![t]\!]_{(k_0, s_1^0 \cdots s_{l_0}^0)}^{\mathcal{A}_3}$

then

(a) for each $j \in [k_0]$: $\lambda_j^0 = |u_j^0|$

(b) for each $j \in [l_0]$ and $\xi_1', \ldots, \xi_{s_j^0}' \in \mathrm{U}_\Delta^*(Y)$:
$\mu_j^0(\theta) = \mathrm{len}(\zeta_j^0[y_o/\xi_o' \mid o \in [s_j^0]])$ if $\theta(y_o) = \mathrm{len}(\xi_o')$ for each $o \in [s_j^0]$, and

(c) for each $j \in [l_0]$, $i \in [s_j^0]$, and $\xi_1', \ldots, \xi_{s_j^0}' \in \mathrm{U}_\Delta^*(Y)$:

$$\xi_i' = \left(\zeta_j^0\left[y_o/\xi_o' \mid o \in [s_j^0]\right]\right)\Big|_{\varrho_j^0(\theta)(i)}^{\to^{\mathrm{len}(\xi_i')}}$$

where $\theta(y_o) = \mathrm{len}(\xi_o')$ for each $o \in [s_j^0]$. □

The next theorem states that the alignment functions α work as supposed. In fact,

(a) each α_j connects valid positions from some string with positions in the j-th hedge (but not positions where variables are located),

(b) each sentence position points to at most one position in some hedge, and

(c) each sentence position points to at least one position in some hedge.

The proof is again in the appendix on pp. 210.

Theorem 4.3.8. Let $t \in (\mathrm{T}_\Sigma)_{(k_0, s_1^0 \cdots s_{l_0}^0)}$ and let

- $(u_1^0, \ldots, u_{k_0}^0) = [\![t]\!]_{k_0}^{\mathcal{A}_1}$,
- $(\zeta_1^0, \ldots, \zeta_{l_0}^0) = [\![t]\!]_{(s_1^0 \cdots s_{l_0}^0)}^{\mathcal{A}_2}$, and
- $(\lambda^0, \mu^0, \varrho^0, \alpha^0) = [\![t]\!]_{(k_0, s_1^0 \cdots s_{l_0}^0)}^{\mathcal{A}_3}$.

For each $j \in [l_0]$ let $\theta_j \colon Y_{s_j^0} \to \mathbb{N}$. The following holds:

(a) For each $j \in [l_0]$ and $\xi'_1, \ldots, \xi'_{s_j^0} \in \mathrm{U}_\Delta^*(Y)$ where, for each $q \in [s_j^0]$, $\theta_j(y_q) = \mathrm{len}(\xi'_q)$, $\alpha_j^0(\theta_j)$ is a partial function with the signature

$$\alpha_j^0(\theta_j) \colon \{(i', j') \mid i' \in [k_0], j' \in [|u_{i'}^0|]\} \dashrightarrow D$$

where

$$D = \mathrm{pos}_\Delta(\zeta_j^0[y_o / \xi'_o]) \setminus \{\varrho_j^0(\theta)(\hat{o}) \cdot p \mid \hat{o} \in [s_j^0], p \in \mathrm{pos}(\xi'_{\hat{o}})\} \ .$$

Also $\alpha_j^0(\theta_j)$ is injective.

(b) For $j_1, j_2 \in [l_0]$ where $j_1 \neq j_2$ we have that

$$\mathrm{dom}(\alpha_{j_1}^0(\theta_{j_1})) \cap \mathrm{dom}(\alpha_{j_2}^0(\theta_{j_2})) = \emptyset \ .$$

(c) $\{(i', j') \mid i' \in [k_0], j' \in [|u_{i'}^0|]\} = \bigcup_{j \in [l_0]} \mathrm{dom}(\alpha_j^0(\theta_j)) \ .$ □

The previous two theorems imply the following corollary. It allows us to define an IRTG that generates hybrid trees in the next section.

Corollary 4.3.9. Let $t \in (\mathrm{T}_\Sigma)_{(1,(0))}$ and

- $(u) = [\![t]\!]_1^{\mathcal{A}_1}$,
- $(\xi) = [\![t]\!]_{(0)}^{\mathcal{A}_2}$, and
- $(\lambda, \mu, \varrho, \alpha) = [\![t]\!]_{(1,(0))}^{\mathcal{A}_3}$.

Then $(u, \xi, \alpha_1(\emptyset))$ is a hybrid tree. □

4.4 LCFRS/sDCP hybrid grammars

Using the preparations of the previous sections we can now define LCFRS/sDCP hybrid grammars. To this end, we just need to combine an LCFRS algebra $\mathcal{A}_1$, an sDCP algebra $\mathcal{A}_2$, an $(\mathcal{A}_1, \mathcal{A}_2)$-alignment algebra $\mathcal{A}_3$, and an RTG G, such that all these ingredients are compatible and choose the start symbol to be of sort 1 with respect to $\mathcal{A}_1$ and sort (0) with respect to $\mathcal{A}_2$. Then by Corollary 4.3.9 we know that the simultaneous interpretation of trees from $L(G)$ in the different algebras yield hybrid trees.

Definition 4.4.1. Let Σ be a ranked alphabet and let Δ and Γ be alphabets. An *LCFRS/sDCP hybrid grammar* is an IRTG $\mathbb{H} = (G, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ where

- $\mathcal{A}_1$ is a $((\Sigma, \mathrm{sort}_{\mathrm{LCFRS}}), \Gamma)$-LCFRS algebra,
- $\mathcal{A}_2$ is a $((\Sigma, \mathrm{sort}_{\mathrm{sDCP}}), \Delta)$-sDCP algebra such that $\mathrm{sort}_{\mathrm{LCFRS}}$ and $\mathrm{sort}_{\mathrm{sDCP}}$ are compatible,
- $G = (N, \Sigma, S, R)$ is a $(\Sigma, \mathrm{sort}_{\mathrm{LCFRS}} \times \mathrm{sort}_{\mathrm{sDCP}})$-compatible RTG,
- $\mathrm{sort}_N(S) = (1, (0))$, and
- $\mathcal{A}_3$ is an $(\mathcal{A}_1, \mathcal{A}_2)$-alignment algebra.

The *hybrid tree language of* $\mathbb{H}$, denoted by $\mathcal{L}(\mathbb{H})$, is defined as follows:

$$\mathcal{L}(\mathbb{H}) = \{(s, \xi, \alpha(\emptyset)) \mid \exists t \in L(G)\colon\ [\![t]\!]_1^{\mathcal{A}_1} = (s), [\![t]\!]_{(0)}^{\mathcal{A}_2} = (\xi), [\![t]\!]_{(1,(0))}^{\mathcal{A}_3} = (\lambda, \mu, \varrho, \alpha)\}\ .$$

□

Example 4.4.2. Let $\Delta = \{a, b, c\}$ and $\Sigma = \{\sigma, \gamma, \delta, \beta\}$ be alphabets. Let $\mathrm{sort}_{\mathrm{LCFRS}}$ and $\mathrm{sort}_{\mathrm{sDCP}}$ be such that

$$\begin{array}{ll}
\mathrm{sort}_{\mathrm{LCFRS}}(\sigma) = (2\ 1, 1) & \mathrm{sort}_{\mathrm{sDCP}}(\sigma) = ((0\ 0)\ (1), (0)) \\
\mathrm{sort}_{\mathrm{LCFRS}}(\gamma) = (2\ 1, 1) & \mathrm{sort}_{\mathrm{sDCP}}(\gamma) = ((0\ 0), (0\ 0)) \\
\mathrm{sort}_{\mathrm{LCFRS}}(\delta) = (\varepsilon, 2) & \mathrm{sort}_{\mathrm{sDCP}}(\delta) = (\varepsilon, (0\ 0)) \\
\mathrm{sort}_{\mathrm{LCFRS}}(\beta) = (\varepsilon, 1) & \mathrm{sort}_{\mathrm{sDCP}}(\beta) = (\varepsilon; (1))
\end{array}$$

Let $\mathcal{A}_1$ be a $((\Sigma, \mathrm{sort}_{\mathrm{LCFRS}}), \Delta)$-LCFRS algebra, $\mathcal{A}_2$ be a $((\Sigma, \mathrm{sort}_{\mathrm{sDCP}}), \Delta)$-sDCP

algebra, and $\mathcal{A}_3$ be an $(\mathcal{A}_1, \mathcal{A}_2)$-alignment algebra such that[2]

$$\sigma^{\mathcal{A}_3} = \left\langle x_1^2 x_1^1 x_2^1 \; ; \; \begin{array}{c} x_1^2 \\ \hline x_1^1 x_2^1 \end{array} \; ; \; \emptyset \right\rangle$$

$$\gamma^{\mathcal{A}_3} = \left\langle x_1^1 a \; , \; a x_2^1 \; ; \; \begin{array}{c} a \\ \hline x_1^1 \end{array} \; , \; \begin{array}{c} a \\ \hline x_2^1 \end{array} \; ; \; \{(1,2) \mapsto (2,\diamond), (2,1) \mapsto (1,\diamond)\} \right\rangle$$

$$\delta^{\mathcal{A}_3} = \left\langle c \; , \; c \; ; \; c \; , \; c \; ; \; \{(1,1) \mapsto (1,\diamond), (2,1) \mapsto (2,\diamond)\} \right\rangle$$

$$\beta^{\mathcal{A}_3} = \left\langle b \; ; \; \begin{array}{c} b \\ \hline y_1 \end{array} \; ; \; \{(1,1) \mapsto (1,\diamond)\} \right\rangle$$

Let $G = ((\{Z, A, B\}, \mathrm{rk}), \Sigma, Z, R)$ be an RTG where $\mathrm{rk}(\sigma) = 2$, $\mathrm{rk}(\gamma) = 1$, $\mathrm{rk}(\delta) = 0$, $\mathrm{rk}(\beta) = 0$, and

$$R = \{ Z \; \to \sigma(A, B) \; , \; A \to \gamma(A) \; , \; A \to \delta() \; , \; B \to \beta() \; \} \; .$$

G is compatible with $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$. The hybrid tree language generated by G is illustrated in Figure 4.6. □

[2] The operations of $\mathcal{A}_1$ and $\mathcal{A}_2$ can be reconstructed from the first and second component of the representations of those of $\mathcal{A}_3$.

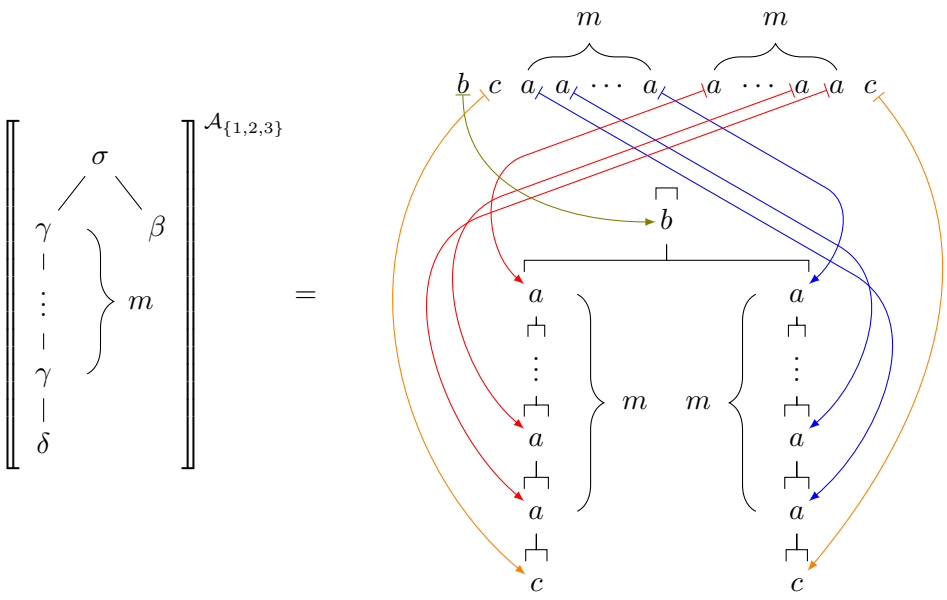


Figure 4.6: Language of a hybrid grammar: a tree from $L(G)$ (left) is interpreted in all algebras. The resulting string (top right) is of form $bca^m a^m c$ ($m \in \mathbb{N}$), the resulting unranked tree (bottom right) of the form $b(a^m(c)\ a^m(c))$, and the alignment is as depicted.

5 Induction of LCFRS/sDCP hybrid grammars

After we defined LCFRS/sDCP hybrid grammars, we now address the question how hybrid grammars can be constructed. Ultimately, we want to obtain a hybrid grammar that can be used for parsing arbitrary sentences w of a particular natural language: its LCFRS component shall recognize w and the evaluation of the parse trees in its sDCP and alignment algebra shall yield hybrid trees that represent the syntactic structure of the sentence. One approach could be to hand-craft a grammar, which requires profound linguistic knowledge of the language at hand, resulting in informed decisions on how to encode certain phenomena in nonterminals and rules. Another approach, which we follow, is to take an extensive collection of syntactically annotated sentences (called corpus) and extract a grammar from it. The linguistic knowledge is now mostly incorporated in the corpus, which is independent of a concrete grammar formalism[1]. The task of the parser creator is then to develop a good algorithm to construct the grammar – naturally, linguistically informed adjustments are beneficial for this task as well.

We describe a method to induce LCFRS/sDCP hybrid grammars from dependency or constituent trees. These methods are slightly generalized and unified versions of the algorithms presented in Gebhardt, Nederhof, and Vogler (2017). We give a single construction for the sDCP part of the grammar that works for either kind of hybrid tree. This construction has a *decomposition* of the hedge positions as input. A decomposition describes how some object is recursively decomposed in smaller parts – the rules of the grammar are chosen to reenact this process. How the decomposition is obtained differs in the dependency and constituent case.

While in Gebhardt, Nederhof, and Vogler (2017) we considered a special kind of decomposition called *recursive partitioning*, our generalized construction also allows to induce hybrid grammars based on arbitrary decompositions of the string positions. In the practical part, we only exploit this property to separate the POS-layer from other syntactic categories (this induction strategy goes beyond those proposed in Gebhardt,

[1] This is true only to the extent that the creation of a corpus is often automated by using a constraint-based parser that infers likely candidates for the human annotator. Also the annotation scheme is usually based on (a not always formalized) grammatical theory.

Nederhof, and Vogler 2017). Moreover, we give an illustration that lexicalized hybrid grammars, in the sense that each rule produces exactly one terminal symbol in the LCFRS algebra, can be induced with this algorithm, if the decomposition is chosen accordingly. An evaluation of this induction strategy is left for future research.

5.1 Decompositions and recursive partitionings

The core idea of the induction algorithm introduced by Nederhof and Vogler (2014) and Gebhardt, Nederhof, and Vogler (2017) is that of a *recursive partitioning*. Intuitively, a recursive partitioning is a tree over subsets of sentence positions that summarized how the individual words are stepwise composed until the whole sentence is obtained. In order to capture discontinuity we just need to allow the subsets of sentence positions to constitute discontinuous intervals.

In this work, we generalize the concept of recursive partitioning to arbitrary sets (and not just sets of sentence positions). Moreover, we loosen the conditions on the parent child relationship. We call the result *decomposition*. This gives on the one hand more general algorithms than those in Gebhardt, Nederhof, and Vogler (2017). On the other hand, it allows us to formalise the connection between the LCFRS and the sDCP that we construct, because we can associate to each decomposition of string positions a decomposition of tree (or hedge) positions.

Definition 5.1.1. Let U be finite set. A U-decomposition is a tree $t \in \mathrm{U}_{\mathfrak{P}(U)}$ such that $U = t(\diamond)$ and, for each $p \in \mathrm{pos}(t)$, the following holds:

$$t(p) \supseteq \bigcup_{p' \in \mathrm{children}^t(p)} t(p') \quad \text{and}$$

$$\emptyset = t(p') \cap (p'') \quad \text{for each } p', p'' \in \mathrm{children}^t(p) \text{ with } p' \neq p'' \ . \qquad \square$$

A recursive partitioning is just a special case of general decompositions.

Definition 5.1.2. Let $n \in \mathbb{N}$. A recursive partitioning for n is an $[n]$-decomposition t where, for each $p \in \mathrm{pos}(t)$, we have that

- $t(p) = \bigcup_{p' \in \mathrm{children}^t(p)} t(p')$ and $\star_p > 1$, or
- $|t(p)| = 1$ and $\mathrm{children}^t(p) = \varepsilon$. $\square$

We recall a few classes (also called *strategies*) of recursive partitionings from Gebhardt, Nederhof, and Vogler (2017). The first two, depicted in Figure 5.1, split greedily away the leftmost or rightmost position, respectively.

Definition 5.1.3. A recursive partitioning t for n is *left-branching* if for each $p \in \mathrm{pos}(t)$ there is $j \in [n]$ such that

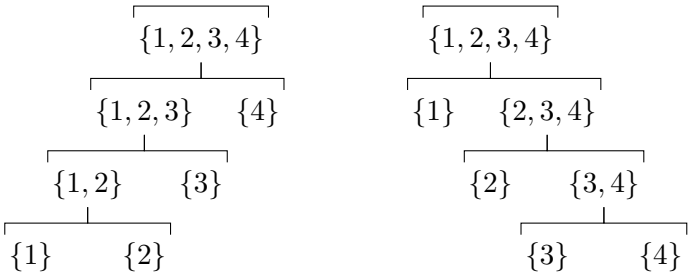


Figure 5.1: The left-branching and the right-branching recursive partitioning for 4.

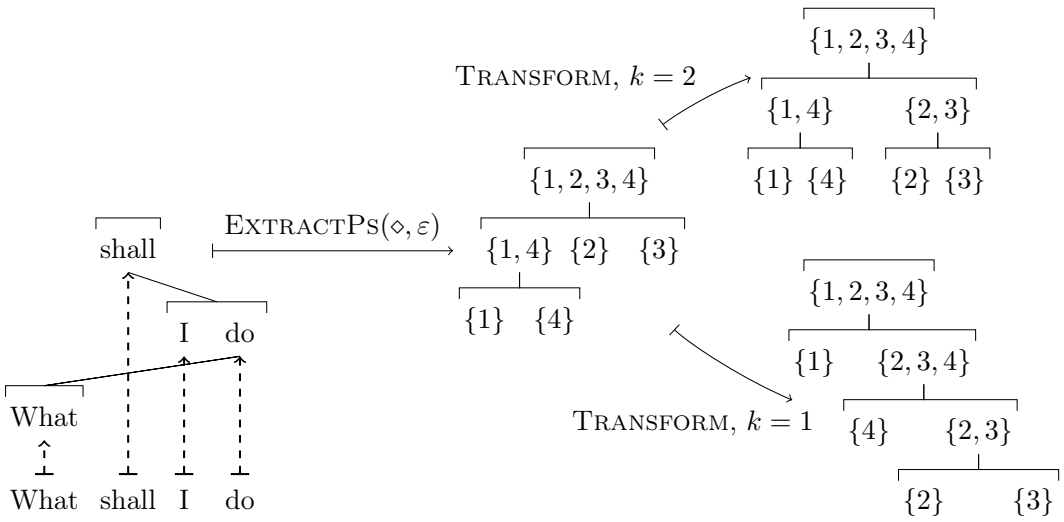


Figure 5.2: Extraction of a recursive partitioning from a dependency structure and subsequent transformation.

- $t|_p = \{1, 2, \ldots, j\}(t|_{p\downarrow\diamond}, \{j\})$ or
- $t|_p = \{j\}$. □

Definition 5.1.4. A recursive partitioning t for n is *right-branching* if for each $p \in \mathrm{pos}(t)$ there is $j \in [n]$ such that

- $t|_p = \{j, j+1, \ldots, n\}(\{j\}, t|_{p\downarrow\rightarrow\diamond})$ or
- $t|_p = \{j\}$. □

Next, we want to extract a recursive partitioning for $|w|$ from a hybrid tree (w, ζ, α) that also takes the hedge ζ into account. One option is to construct a $[|w|]$-decomposition t with $\mathrm{pos}(t) = \mathrm{pos}(\zeta)$. For each position p in $\mathrm{pos}(\zeta)$, $t(p)$ is set to $\{i \in [|w|] \mid \exists k \in \mathbb{N}\colon \mathrm{parent}^k(\alpha(i)) = p\}$, i.e., the sentence positions that are aligned

Algorithm 5.1.1 Algorithm to extract a recursive partitioning from a hybrid tree.

Require: hybrid tree $h = (w, \zeta, \alpha)$, $p \in \mathrm{pos}(\zeta)$
Ensure: $\textsc{ExtractP}^h(p) = t$ such that
$t = \varepsilon$ if $\mathrm{cover}^h(p) = \emptyset$ and $t \in \mathrm{U}_{\mathfrak{P}([|w|])}$ with $t(\diamond) = \mathrm{cover}^h(p)$ otherwise
function $\textsc{ExtractP}^h(p)$
  if $\exists i \in [|w|] \colon \alpha(i) = p$ **then**
    return $\textsc{ExtractPs}^h(\mathrm{children}^\zeta(p), \{i\})$
  else
    return $\textsc{ExtractPs}^h(\mathrm{children}^\zeta(p), \varepsilon)$

Require: hybrid tree $h = (w, \zeta, \alpha)$, $P \in \mathrm{pos}(\zeta)^*$, $s \in \mathrm{U}^*_{\mathfrak{P}([|w|])}$
let $C = \bigcup_{p \in P} \mathrm{cover}^h(p) \cup \bigcup_{i \in [|s|]} s(i)$
Ensure: $\textsc{ExtractPs}^h(P, s) = t$ such that
$t = \varepsilon$ if $C = \emptyset$ and $t \in \mathrm{U}_{\mathfrak{P}([|w|])}$ is a decomposition of C otherwise
function $\textsc{ExtractPs}^h(P,\ s)$
  for $p' \in P$ **do**
    $s \leftarrow s \cdot \textsc{ExtractP}^h(p')$
  if $\mathrm{len}(s) \leq 1$ **then return** s
  else
    $U = \bigcup_{i=0}^{\mathrm{len}(s)-1} s(\rightarrow^i \diamond)$
    sort s such that $\min(s(\rightarrow^j \diamond)) < \min(s(\rightarrow^{j+1} \diamond))$
    return $U(s)$

to p or a descendant of p. In general, t is not yet a recursive partitioning. Therefore, Algorithm 5.1.1 collapses chains of the form $U(U(\ldots))$, adds missing singletons, and also orders the children in t by the smallest sentence position they contain.

Definition 5.1.5. Let $h = (w, \zeta, \alpha)$ be a hybrid tree. The *recursive partitioning directly extracted from* h is obtained by calling

$$\textsc{ExtractPs}^h((\rightarrow^0 \diamond) \cdots (\rightarrow^{\mathrm{len}(\zeta)-1} \diamond), \varepsilon)$$

from Algorithm 5.1.1. □

An example for the extraction of the recursive partitioning is given in Figure 5.2 on the left. Intuitively, the directly extracted recursive partitioning describes a decomposition process very similar to the vanilla, unbinarized LCFRS that one would induce from the same hybrid tree[2]. In particular, this implies that the degree of discontinuity in the hybrid tree is preserved in the recursive partitioning extracted from it.

[2]See the induction algorithms in Maier and Søgaard (2008) and Kuhlmann (2013).

5.2 Inducing an LCFRS from a string and a decomposition

In this section, we describe an approach to obtain LCFRSs from hybrid trees. In preparation we define a few notions that will be helpful for this purpose.

Definition 5.2.1. Let $U \subseteq \mathbb{N}$ be a finite set. We define the *fanout of* U as the smallest number k such that U can be partitioned into sets $U_1, \ldots, U_k$ where

- for each $i \in [k]$, there is $\ell \in \mathbb{N}$ such that $U_i = \{\ell + 1, \ldots, \ell + |U_i|\}$, and
- for each $i \in [k-1]$, $\min(U_i) < \min(U_{i+1})$.

We denote $(U_1, \ldots, U_k)$ by $\mathrm{span}(U)$.
The notion of *fanout* can be generalized to U-decompositions: Let t be a U-decomposition. We define the fanout of t as $\max_{p \in \mathrm{pos}(t)} \mathrm{fanout}(t(p))$. □

From a decomposition of the positions of some string, an LCFRS algebra can be constructed.

Definition 5.2.2. Let $w \in \Delta^*$ and t be a $[|w|]$-decomposition. We define the $\mathbb{N}^* \times \mathbb{N}$-sorted alphabet $\Sigma^{w,t} = \{\sigma_{p_0} \mid p_0 \in \mathrm{pos}(t)\}$, where, for each $p_0 \in \mathrm{pos}(t)$, σ_{p_0} is a fresh symbol. Let $p_0 \in \mathrm{pos}(t)$ and $\mathrm{children}^t(p_0) = p_1 \cdots p_n$. For each $j \in [n]_0$, let $\mathrm{span}(t(p_i)) = (U_1^i, \ldots, U_{k_i}^i)$. We set $\mathrm{sort}_{\Sigma^{w,t}}(\sigma_{p_0}) = (k_1 \cdots k_n, k_0)$. We define f_{p_0} to be the function

$$\langle u_1, \ldots, u_{k_0} \rangle \colon (\Delta^*)^{k_1} \times \cdots \times (\Delta^*)^{k_n} \to (\Delta^*)^{k_0} \ ,$$

where each $u_j = u_j^1 \cdots u_j^m$ $(j \in [k_0])$ is constructed as follows: Partition U_j^0 into sets $V_1, \ldots, V_m$ where, for each $\ell \in [m-1]$, $\min(V_\ell) < \min(V_{\ell+1})$ and, for each $\ell \in [m]$,

(i) either $V_\ell = U_{i'}^{j'}$ for some $j' \in [n]$ and $i' \in [k_{j'}]$ – in this case $u_j^\ell = x_{i'}^{j'}$;

(ii) or $V_\ell = \{q_\ell\}$ with $q_\ell \in U_j^0 \setminus \left(\bigcup_{j' \in [n]} t(p_{j'}) \right)$ – in this case $u_j^\ell = w[q_\ell]$. In this case we also say that $q_\ell \sim_{p_0} (j, \ell)$.

We define the $(\Sigma^{w,t}, \Delta)$-LCFRS algebra $\mathcal{A}^{w,t}$ where $\sigma_{p_0}^{\mathcal{A}^{w,t}} = f_{p_0}$ for each $p_0 \in \mathrm{pos}(t)$. □

The constructed LCFRS algebra has the natural property to generate the string as outlined by the decomposition. This is formalized in the next observation.

Observation 5.2.3. Let $w \in \Delta^*$, t be a $[|w|]$-decomposition, and $\mathcal{A}^{w,t}$ as in Definition 5.2.2. For each $p \in \mathrm{pos}(t)$ with $\mathrm{children}^t(p) = p_1 \cdots p_n$ and fanout of $t(p)$ being k, define $\xi_p \in (\mathrm{T}_{\Sigma^{w,t}})_k$ recursively such that

$$\xi_p = \sigma_p(\xi_{p_1}, \ldots, \xi_{p_n}) \ .$$

Then $[\![\xi_p]\!]_k^{\mathcal{A}^{w,t}} = (w[\ell_1 : r_1], \ldots, w[\ell_k : r_k])$ where

$$\mathrm{span}(t(p)) = (\{\ell_1 + 1, \ldots, r_1\}, \ldots, \{l_k + 1, \ldots, r_k\}) \ . \qquad \square$$

Once we obtained a recursive partitioning t for $|w|$ from a hybrid tree (w, ζ, α), we can leverage Definition 5.2.2 to construct an LCFRS algebra because t is a particular instance of a $[|w|]$-decomposition. Using the labels of t as nonterminal symbols and their parent-child relationship in t as blueprints for rules, we define an LCFRS that generates w.

Definition 5.2.4. Let w be a string and t be a $[|w|]$-decomposition. In particular, t might be a recursive partitioning for $|w|$. We construct the LCFRS $\mathbb{G}^{w,t} = (G^{w,t}, \mathcal{A}^{w,t})$ with $\mathcal{A}^{w,t}$ as in Definition 5.2.2 and $G^{w,t} = (N, \Sigma^{w,t}, S, R)$ such that

- $N = \{t(p) \mid p \in \mathrm{pos}(t)\}$ where $\mathrm{sort}_N(t(p))$ is the fanout of $t(p)$,
- $S = [|w|]$, and
- $R = \{t(p\diamond) \to \sigma_{p\diamond}\left(\ t(p \downarrow \to^0 \diamond) \ , \ldots, \ t(p \downarrow \to^{\star_{p\diamond}-1} \diamond) \ \right) \mid p\diamond \in \mathrm{pos}(t)\}$. $\square$

One can show the following proposition (proof omitted).

Proposition 5.2.5. $\{w\} = \mathcal{L}(G^{w,t}, \mathcal{A}^{w,t})$. $\square$

Example 5.2.6. Let w be the sentence fragment from the German TiGer corpus

„	Es	bestünde	somit	hinreichend	Spielraum	“
(	There	would be	therefore	sufficient	room for manoeuvre	)

where the alphabet Δ is over German words.

Consider the decomposition $t = V_0(V_1(V_2(V_3(V_4))))$ of $[|w|]$, where, for each $i \in [4]_0$, $V_i = \{i + 1, \ldots, 5\}$. We obtain the following LCFRS $\mathbb{G}^{w,t} = (G^{w,t}, \mathcal{A}^{w,t})$ where $G^{w,t} = (\{V_0, \ldots, V_4\}, \Sigma^{w,t}, V_0, R)$ and the rules in R and operations of $\mathcal{A}^{w,t}$ are as follows:

$$\begin{aligned}
V_0 &\to \sigma_\diamond(V_1) & \sigma_\diamond^{\mathcal{A}^{w,t}} &= \langle \text{Es } x_1^1 \rangle \\
V_1 &\to \sigma_{\downarrow\diamond}(V_2) & \sigma_{\downarrow\diamond}^{\mathcal{A}^{w,t}} &= \langle \text{bestünde } x_1^1 \rangle \\
V_2 &\to \sigma_{\downarrow^2\diamond}(V_3) & \sigma_{\downarrow^2\diamond}^{\mathcal{A}^{w,t}} &= \langle \text{somit } x_1^1 \rangle \\
V_3 &\to \sigma_{\downarrow^3\diamond}(V_4) & \sigma_{\downarrow^3\diamond}^{\mathcal{A}^{w,t}} &= \langle \text{hinreichend } x_1^1 \rangle \\
V_4 &\to \sigma_{\downarrow^4\diamond}() & \sigma_{\downarrow^4\diamond}^{\mathcal{A}^{w,t}} &= \langle \text{Spielraum} \rangle
\end{aligned}$$

We observe that $\mathbb{G}^{w,t}$ does not exploit the potential of LCFRS because it is a syntactic variant of a Chomsky Type-3 grammar. $\square$

Algorithm 5.2.1 Transformation of recursive partitioning

Require: a recursive partitioning π of n, an integer $k \geq 1$
Ensure: a binary recursive partitioning π' of fanout no greater than k

function TRANSFORM($\pi = J(t_1 \dots t_m)$)
  if $|J| = 1$ **then**
    return $J()$
  breadth-first search p in $\mathrm{pos}(\pi) \setminus \{\diamond\}$ such that $\pi(p)$ and $J \setminus \pi(p)$ have fanout $\leq k$
  $t \leftarrow$ FILTER($\pi(p), \pi$)
  return $J($TRANSFORM($\pi|_p$) TRANSFORM(t)$)$
function FILTER(J', $\pi = J(t_1 \dots t_m)$) ▷ $J' \subseteq [n]$, $\pi \in \mathrm{U}_{\mathfrak{P}([n])}$
  $F \leftarrow J \setminus J'$
  if $|F| = 1$ **then** **return** $F()$
  else if $|F| = 0$ **then return** ε
  else
    $s \leftarrow$ FILTER(J', t_1) $\cdot \ldots \cdot$ FILTER(J', t_m)
    if $|s| = 1$ **then** **return** s
    else **return** $F(s)$

From Observation 5.2.3 we know that the fanout of the recursive partitioning determines the fanout of the LCFRS algebra that we obtain from it. Nederhof and Vogler (2014) introduced hybrid grammars as a means to generate discontinuous structures while controlling the fanout and, in consequence, the parsing complexity. They described an algorithm, later formalized in Gebhardt, Nederhof, and Vogler (2017), which transforms a given recursive partitioning. The resulting recursive partitioning is binary, i.e., each node has at most two children, and each node has a fanout below a target value $k \in \mathbb{N}_+$. A variant of this algorithm adopted to the notation of this thesis is given as Algorithm 5.2.1 and two example transformation are given in Figure 5.2. The central idea is to search a node p in the given recursive partitioning π such that both $\pi(p)$ and $\pi(\diamond) \setminus \pi(p)$ satisfy the fanout target. Such a node p must exist if $\pi(\diamond)$ has fanout $\leq k$, because one can always choose the position labeled by the singleton containing the minimum (or maximum) of $\pi(\diamond)$. The precondition is true for the root of any recursive partitioning because it is of the form $[n]$, i.e., it has fanout one. The new partitioning has $\pi(\diamond)$ as root. Its first child is the transformed version of $\pi|_p$. Its second child is the transformed version of π where all positions in $\pi(p)$ have been filtered out. Again we know that the inputs to the recursive calls of the transformation algorithm satisfy the fanout target at the root, because this was exactly how p was chosen.

5.3 Stenciling unranked hedges

Our next goal is to construct an sDCP algebra from a given hedge ξ and a decomposition of its positions. In preparation, we define a so-called *stencil operation* that cuts out subhedges located at certain spans from ξ and replaces them by variables. The resulting context, which we regard as a stencil, can afterward be used to define the operations of the algebra. For instance, in the hedge $\alpha\beta\gamma(\delta)$ we may cut out the span $(\varepsilon, \to)$, which includes α, and the span $(\to^2\downarrow, \to)$, which corresponds to δ. As a result we obtain the context $y_1\beta\gamma(y_2)$.

For the stencil operation to be well-defined, we require that the subhedges which we cut out do not overlap. Since variables in a context occur in lexicographic order by definition, the spans that we consider need to adhere to this order as well.

Definition 5.3.1. Let $\xi \in \mathrm{U}^*_\Delta$. Let $k \in \mathbb{N}$ and $(w_1, \to^{i_1}), \ldots, (w_k, \to^{i_k})$ be such that,

(a) for each $o \in [k]$, $(w_o, \to^{i_o})$ is a span position pair,

(b) for each $o, o' \in [k]$ with $o \neq o'$, $(w_o, \to^{i_o})$ and $(w_{o'}, \to^{i_{o'}})$ are in parallel, and

(c) $o < o'$ implies $w_o \to^{i_o} \preceq_{\mathrm{lex}} w_{o'} \to^{i_{o'}}$, where $\preceq_{\mathrm{lex}}$ is the lexicographic order induced by $\downarrow \prec \to$.

We call $(w_1, \to^{i_1}), \ldots, (w_k, \to^{i_k})$ *ordered parallel span position pairs for* ξ. □

Next, we define the *stencil operation*. Note that it is possible to cut out the same empty span multiple times: e.g., cutting out $(\to, \to^0)$ three times and the span $(\to, \to^1)$ once in the hedge $\alpha\beta\gamma$ results in $\alpha y_1 y_2 y_3 y_4 \gamma$ where y_1, y_2, y_3 correspond to the empty spans whereas y_4 corresponds to the last span. This aspect is reflected in items 2 and 3 of the definition.

Definition 5.3.2. Let $\xi \in \mathrm{U}^*_\Delta$ and let $(w_1, \to^{i_1}), \ldots, (w_k, \to^{i_k})$ be ordered parallel span position pairs for ξ. Note that, for each $o \in [k]$, w_o is of the form $\to^{j_o} w'_o$ where $j_o \in \mathbb{N}$ and $w'_o = \varepsilon$ or $w'_o \in \{\downarrow\} \cdot \{\to, \downarrow\}^*$.

The *stencil operation applied to* ξ *at* $(w_1, \to^{i_1}), \ldots, (w_k, \to^{i_k})$, denoted by

$$\xi \ltimes \{(w_o, \to^{i_o}) \mapsto y_o \mid o \in [k]\} \ ,$$

yields the context $\zeta \in \mathrm{C}^*_\Delta(Y_k)$ defined recursively as follows:

1. Let $n \in \mathbb{N}$ and $\xi_1, \ldots, \xi_n \in \mathrm{U}_\Delta$ such that $\xi = \xi_1 \cdots \xi_n$.
2. $[k]$ is partitioned into sets $O_1, \ldots, O_\kappa$ where
   - for each $o, o' \in [k]$, $\exists \ell \in [\kappa]$ such that $\{o, o'\} \subseteq O_\ell$ if and only if $j_o = j_{o'}$ (hence, we define $\hat{j}_\ell = j_o$ for an arbitrary representative $o \in O_\ell$) and
   - $1 \leq \ell < \ell' \leq \kappa$ if and only if $\hat{j}_l < \hat{j}_{\ell'}$.

3. For each $\ell \in [\kappa]$, let $O_\ell = \{o_1, \ldots, o_{\kappa'}\}$ with $o_x < o_{x+1}$ for each $x \in [\kappa' - 1]$. We define v_ℓ as follows:

$$v_\ell = v_\ell^1 \cdots v_\ell^{\kappa'} \zeta_\ell$$

$$\text{where} \quad v_\ell^x = \begin{cases} y_{o_x} & \text{if } w'_{o_x} = \varepsilon \\ \varepsilon & \text{otherwise} \end{cases}$$

$$\zeta_\ell = \begin{cases} \sigma(\xi' \ltimes Q) & \text{if } \xi_{\hat{j}_\ell+1} = \sigma(\xi') \text{ and } \exists o \in O_\ell \colon w'_o =\downarrow w''_o \\ & Q = \{(w''_o, \rightarrow_{i_o}) \mapsto y_o \mid o \in O_\ell \colon w'_o =\downarrow w''_o\} \\ \varepsilon & \text{otherwise} \end{cases}$$

4. For each $\ell \in [\kappa]_0$ we let $u_\ell = \xi_{\hat{i}_\ell+1} \cdots \xi_{\hat{j}_{\ell+1}}$ with $\hat{i}_0 = 1$,

$$\hat{i}_\ell = \hat{j}_\ell + \max(\{1 \mid \exists o \in O_\ell \colon w'_o \neq \varepsilon\} \cup \{i_o \mid o \in O_\ell \colon w'_o = \varepsilon\})$$

   if $\ell > 0$, and $\hat{j}_{\kappa+1} = n$.

5. Then $\zeta = u_0 v_1 u_1 \cdots u_{\kappa-1} v_\kappa u_\kappa$. □

We show that the stencil operation is well defined.

Lemma 5.3.3. Let ξ and $(w_1, \rightarrow^{i_1}), \ldots, (w_k, \rightarrow^{i_k})$ be as in Definition 5.3.2. Then:

$$\left(\xi \ltimes \{(w_o, \rightarrow^{i_o}) \mapsto y_o \mid o \in [k]\}\right)[y_o/\xi|_{w_o}^{\rightarrow^{i_o}} \mid o \in [k]] = \xi \ . \qquad \square$$

A proof is given in the Appendix (pp. 212).

The stencil operation applied at ordered parallel span position pairs cuts holes into a hedge *from below*. We also need to restrict a hedge *from above*. For this, we can simply use the subhedge operation $\xi|_{w_0}^{\rightarrow^i}$ applied at a span position pair $(w_0, \rightarrow^{i_0})$. We call the combination of $(w_0, \rightarrow^{i_0})$ with ordered parallel span position pairs below $(w_0, \rightarrow^{i_0})$ a *stencil boundary*.

Definition 5.3.4. Let $\xi \in \mathrm{U}_\Delta^*$. Let $(w_0, \rightarrow^{i_0})$ be a span position pair for ξ, and let $(w_1, \rightarrow^{i_1}), \ldots, (w_k, \rightarrow^{i_k})$ be ordered parallel span position pairs for $\xi|_{w_0}^{\rightarrow^{i_0}}$. We say that $((w_1, \rightarrow^{i_1}) \cdots (w_k, \rightarrow^{i_k}), (w_0, \rightarrow^{i_0}))$ *is a stencil boundary for* ξ *with* k *gaps*. We denote the set of *stencil boundaries of* ξ *with* k *gaps* by $\mathrm{SB}_k^{\ltimes}(\xi)$.

We say that $((w_1, \rightarrow^{i_1}) \cdots (w_k, \rightarrow^{i_k}), (w_0, \rightarrow^{i_0}))$ is *concise* if, for each $j \in [k]$, we have that $w_j \notin \{\rightarrow\}^*$ and $i_j > 0$ and, for each $j, j' \in [k]$, we have that $w_j \rightarrow^j \neq w_{j'}$. □

Intuitively, a stencil boundary is concise if there are no holes at root level, each hole has length at least 1, and holes do not touch.

Given a stencil boundary for a hedge ξ we can define the position of ξ that are within the boundary.

Definition 5.3.5. Let $\xi \in \mathrm{U}_{\Delta}^{*}$ and let $w = ((w_1, \to^{i_1}) \cdots (w_k, \to^{i_k}), (w_0, \to^{i_0}))$ in $\mathrm{SB}_k^{\ltimes}(\xi)$. The *set of positions spanned by* w, denoted by $\mathrm{hspan}^{\xi}(w)$, contains each position $p \in \mathrm{pos}(\xi)$ where

- $p = w_0 w' \diamond$ with $w' \in \{\downarrow, \to\}^*$ and $\to^{i_0}$ is not a prefix of w', and
- if $j \in [k]$ is such that $w_0 w_j$ is a prefix of p, then also $w_0 w_j \to^{i_j}$ is a prefix of p. □

Similar to span position pairs we define a few relations on stencil boundaries depending on their relative location to another.

Definition 5.3.6. Let $\xi \in \mathrm{U}_{\Delta}^{*}$, $k, \ell \in \mathbb{N}$, $w = ((w_1, \to^{i_1}) \cdots (w_k, \to^{i_k}), (w_0, \to^{i_0}))$ be in $\mathrm{SB}_k^{\ltimes}(\xi)$, and $u = ((u_1, \to^{j_1}) \cdots (u_\ell, \to^{j_\ell}), (u_0, \to^{j_0}))$ be in $\mathrm{SB}_\ell^{\ltimes}(\xi)$.

- We say that w *encompasses* u if
  - $(u_0, \to^{j_0}) \sqsubseteq (w_0, \to^{i_0})$,
  - there is no $m \in [k]$ such that $(u_0, \to^{j_0})$ and $(w_0 w_m, \to^{i_m})$ are crossing, and,
  - for each $m \in [k]$, if $(w_0 w_m, \to^{i_m}) \sqsubseteq (u_0, \to^{j_0})$, then there is $n \in [\ell]$ such that $(w_0 w_m, \to^{i_m}) \sqsubseteq (u_0 u_n, \to^{j_n})$.
- We say that w *is below* u, denoted by $w \sqsubseteq u$, if there is $m \in [\ell]$ such that $(w_0, \to^{i_0}) \sqsubseteq (u_0 u_\ell, \to^{j_\ell})$.
- We say that w and u are *in parallel*, if $(w_0, \to^{i_0})$ and $(u_0, \to^{j_0})$ are in parallel.
- We say that w and u are *non-crossing*, if w encompasses u, u encompasses w, w is below u, u is below w, or w and u are in parallel.
- We say that w and u are *non-overlapping*, if w is below u, u is below w, or w and u are in parallel. □

5.4 Construction of sDCP algebras and sDCPs

We are prepared to described our method for the construction of sDCP. This method is very similar to that of the construction of LCFRS:

- We assume a hedge ζ and a $\mathrm{pos}(\zeta)$-decomposition t as input to our construction. This pair of inputs corresponds to the string w and the $[|w|]$-decomposition that we required for LCFRS.
- For each position in t labeled with $U \subseteq \mathrm{pos}(\zeta)$, we identify a minimal set B of connected subhedges (cf. Definition 5.4.1). We call the stencil boundaries that delimit the subhedges in B *boundary of* U and refer to arity of B as $\mathrm{srk}(U)$. This is in analogy to the concepts of span and fanout defined in Definition 5.2.1.

- Then for each position p in t labeled with U we determine how the subhedges delimited by boundary(U) can be decomposed into the subhedges already captured by p's children and additional symbols. This process is described by Algorithm 5.4.1. Intuitively, for each subhedge ζ_q delimited by boundary(U), we construct a hedge context ξ_q. For each subhedge of ζ_q that is realized already as the j-th subhedge of the i-th child of p, we insert a second-order variable x_j^i into ξ_q. For each hole in ζ', i.e., a position of ζ_q that is not in U, we insert a first-order variable into ξ_q. Any remaining symbol in ζ_q is realized directly in ξ_q. Finally all the hedge contexts are grouped to an operation $\langle \xi_1, \ldots, \xi_{\mathrm{srk}(U)} \rangle$ of our sDCP algebra.

  Note again that this process is very similar to the construction of operations for the LCFRS algebra, where we inserted variables for string-ranges already realized by one of the children and realized other symbols directly.

We start by giving the formal definition of boundary and srk. An abstract example is given in Table 5.3.

Definition 5.4.1. Let $\zeta \in \mathrm{U}_\Delta^*$ and $U \subseteq \mathrm{pos}(\zeta)$. We define $\mathrm{srk}(U)$ to be the smallest number l such that there are pairwise non-overlapping concise stencil boundaries $w_1, \ldots, w_l$ for ζ where

- for each $i, j \in [l]$ with $i \neq j$, $\mathrm{hspan}^\zeta(w_i) \cap \mathrm{hspan}^\zeta(w_j) = \emptyset$,
- $U = \bigcup_{i \in [l]} \mathrm{hspan}^\zeta(w_i)$, and
- for each $0 < i < j \leq l$ where $w_i = (w_i', (p_i, \to^{k_i}))$ and $w_j = (w_j', (p_j, \to^{k_j}))$, it holds that $p_i \preceq_{\mathrm{lex}} p_j$.

We denote $\mathrm{boundary}^\zeta(U) = (w_1, \ldots, w_l)$ and $\bar{s}_U = s_1 \ldots s_l$ where, for each $i \in [l]$, $s_i \in \mathbb{N}$ is such that $w_i = ((p_1, \to^{j_1}) \cdots (p_{s_i}, \to^{j_{s_i}}), (p_0, \to^{j_0}))$. We define

$$\top^\zeta(U) = \{p \in U \mid \mathrm{parent}(p) \notin U\} \qquad \text{and}$$
$$\bot^\zeta(U) = \{p \in \mathrm{pos}(\zeta) \setminus U \mid \mathrm{parent}(p) \in U\} \ . \qquad \square$$

Observation 5.4.2. Let $\zeta \in \mathrm{U}_\Delta^*$ and $U \subseteq \mathrm{pos}(\zeta)$. Then $\mathrm{boundary}^\zeta(U)$ is unique and therefore well-defined. This follows from the conciseness of $w_1, \ldots, w_l$. $\square$

Definition 5.4.3. Let $\zeta \in \mathrm{U}_\Delta^*$ and t be a $\mathrm{pos}(\zeta)$-decomposition. Let $p_0 \in \mathrm{pos}(t)$ and $\mathrm{children}^t(p_0) = p_1 \cdots p_n$. For each $i \in [n]_0$, let

$$U_i = t(p_i) \qquad \text{and} \qquad \mathrm{boundary}^\zeta(U_i) = (w_i^1, \ldots, w_i^{\mathrm{srk}(U_i)}) \ .$$

For each $q \in [\mathrm{srk}(U_0)]$, the function $\textsc{FillRec}_q$ is defined in Algorithm 5.4.1. The function constructs a hedge contexts with second-order variables. Additionally,

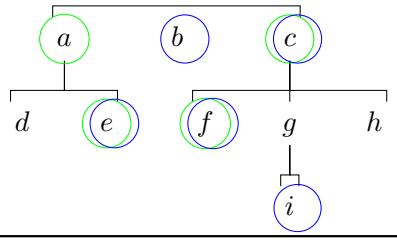
ζ	$\{\zeta(p) \mid p \in U\}$	srk(U)	boundary$^{\zeta}(U)$
	$\{a, c, e, f\}$ (green)	2	$((\downarrow, \rightarrow), (\varepsilon, \rightarrow))$ $((\downarrow\rightarrow, \rightarrow^2), (\rightarrow^2, \rightarrow))$
	$\{b, c, e, f, i\}$ (blue)	3	$(\varepsilon, (\downarrow\rightarrow, \rightarrow))$ $((\rightarrow\downarrow\rightarrow, \rightarrow^2), (\rightarrow, \rightarrow^2))$ $(\varepsilon, (\rightarrow^2\downarrow\rightarrow\downarrow, \rightarrow))$

Table 5.3: A hedge ζ and two examples for a set U, its srk, and the corresponding minimal sequence of concise stencil boundaries.

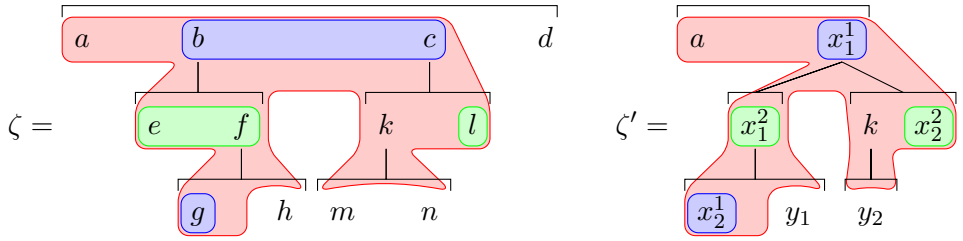


Figure 5.4: An unranked hedge ζ, a set of positions U_0 indicated in red, and sets of positions U_1 and U_2 indicated in blue and green, respectively, that are subsets of U_0. The hedge-context ζ' with second-order variables extracted from ζ by Algorithm 5.4.1.

FillRec$_q$ also populates the initially empty, partial function lookup$_q$ during its execution. We define $f[\zeta, U_0, U_1, \ldots, U_n] = \langle \xi_1, \ldots, \xi_{\text{srk}(U_0)} \rangle$ where, for each $q \in [\text{srk}(U_0)]$, we let $\xi_q = \text{FillRec}_q(p, \rightarrow^i, \varepsilon)$ with p and i such that $w_0^q = (w', (p, \rightarrow^i))$. □

We illustrate Algorithm 5.4.1 by an abstract example.

Example 5.4.4. Consider the unranked hedge ζ depicted in Figure 5.4 and the sets of positions U_0, U_1, and U_2 uniquely defined by the labels of ζ in these positions:

$$\{\zeta(p) \mid p \in U_0\} = \{a, b, c, e, f, g, k, l\}$$
$$\{\zeta(p) \mid p \in U_1\} = \{b, c, g\}$$
$$\{\zeta(p) \mid p \in U_2\} = \{e, f, k, l\}$$

Algorithm 5.4.1 Construction of a hedge with second-order variables

Require: $\zeta \in \mathrm{U}^*_\Delta$, $U_{i'} \subseteq \mathrm{pos}(\zeta)$, $\mathrm{boundary}^\zeta(U_{i'}) = (w^1_{i'}, \dots, w^{\mathrm{srk}(w_{i'})}_{i'})$, $q \in [\mathrm{srk}(U_0)]$, $(p, \to^i)$ span position pair of ζ, $\tilde{p} \in \{\to, \downarrow\}^*$, $\mathrm{lookup}_q \colon \mathrm{pos}(\zeta) \dashrightarrow \{\to, \downarrow, \searrow, *, \diamond\}^*$
Ensure: $\textsc{FillRec}_q(p, \to^i, \tilde{p}) \in \mathrm{U}^*_\Delta(Y, X)$

1: **function** $\textsc{FillRec}_q(p, \to^i, \tilde{p})$
2: **if** $i = 0$ **then**
3: **return** ε
4: **if** $\exists i' \in [n], j' \in [\mathrm{srk}(U_{i'})] \colon p\diamond \in \mathrm{hspan}^\zeta(w^{j'}_{i'})$ **then**
5: let $((p_1, \to^{i_1}) \dots (p_m, \to^{i_m}), (p_0, \to^{i_0})) = w^{j'}_{i'}$
6: **return** $x^{i'}_{j'}(\textsc{FillRec}(p_0p_1, \to^{i_1}, \tilde{p}\,{}_1\!\searrow), \dots, \textsc{FillRec}(p_0p_m, \to^{i_m}, \tilde{p}\,{}_{i_m}\!\searrow)) \cdot \textsc{FillRec}(p \to^{i_0}, \to^{i-i_0}, \tilde{p} \to)$
7: **if** $p\diamond \in \mathrm{hspan}^\zeta(w^q_0)$ **then**
8: $\mathrm{lookup}_q(p\diamond) \leftarrow \tilde{p}\diamond$
9: **return** $\zeta(p\diamond)(\textsc{FillRec}_q(p\downarrow, \to^{\star_{p\diamond}}, \tilde{p}\downarrow)) \cdot \textsc{FillRec}_q(p \to, \to^{i-1}, \tilde{p} \to)$
10: **else** ▷ $p\diamond \in \bot^\zeta(\mathrm{hspan}^\zeta(w^q_0))$
11: let $((p_1, \to^{i_1}) \dots (p_m, \to^{i_m}), (p_0, \to^{i_0})) = w^q_0$
12: let $o \in [m]$ such that $p = p_0p_o$
13: **return** $y_o \cdot \textsc{FillRec}_q(p \to^{i_o}, \to^{i-i_o}, \tilde{p} \to)$

Notably, $U_1 \subseteq U_0 \supseteq U_2$.

$$\mathrm{boundary}^\zeta(U_0) = \Big(\underbrace{((\to\downarrow\to\downarrow\to, \to)(\to^2\downarrow^2, \to^2), (\varepsilon, \to^3))}_{w^1_0}\Big)$$

$$\mathrm{boundary}^\zeta(U_1) = \Big(\underbrace{((\downarrow, \to^2)(\to\downarrow, \to^2), (\to, \to^2))}_{w^1_1}, \underbrace{(\varepsilon, (\to\downarrow\to\downarrow, \to))}_{w^2_1}\Big)$$

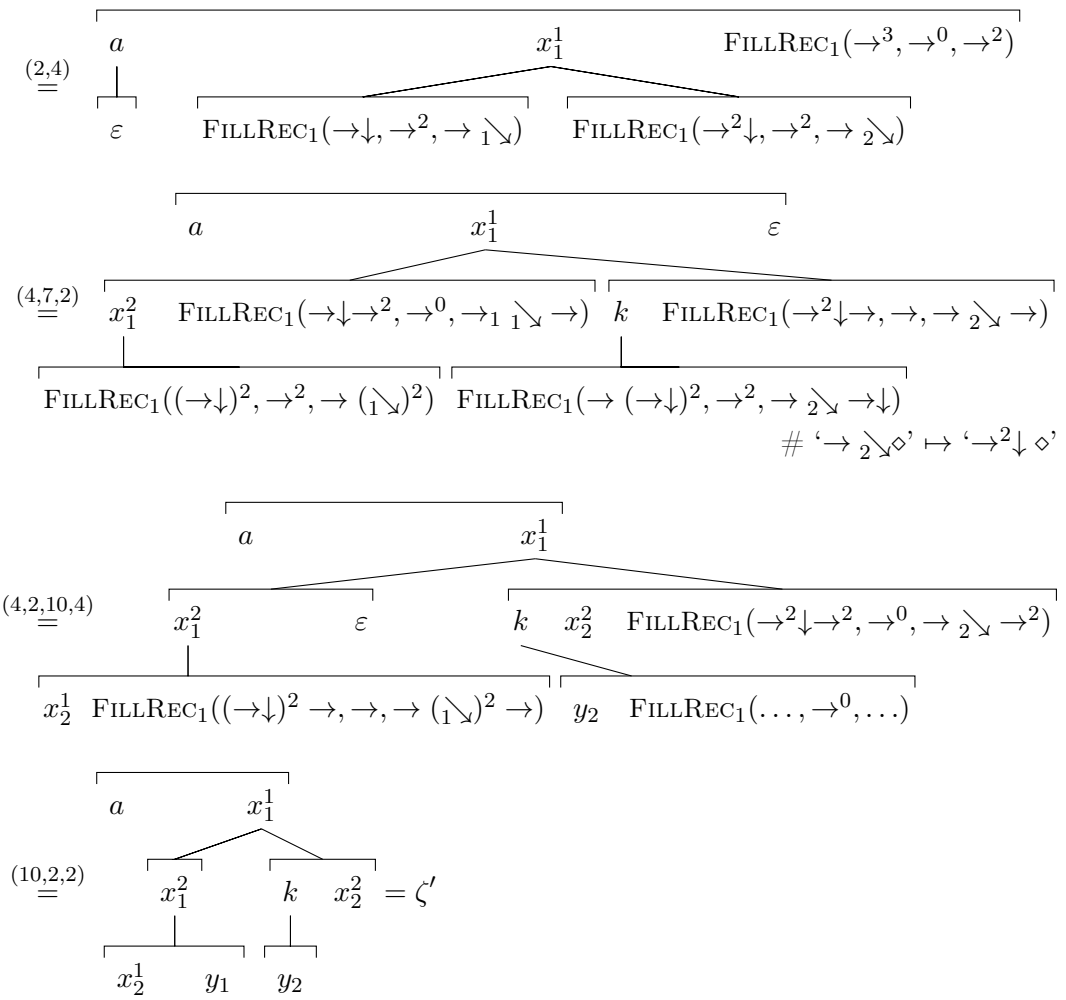
$$\mathrm{boundary}^\zeta(U_2) = \Big(\underbrace{((\to\downarrow, \to^2), (\to, \downarrow^2))}_{w^1_2}, \underbrace{(\varepsilon, (\to^2\downarrow\to, \to))}_{w^2_2}\Big)$$

The extraction of the context ζ' from ζ by executing Algorithm 5.4.1 is documented below. Additions to lookup_1 are documented after a #.

$\textsc{FillRec}_1(\varepsilon, \to^3, \varepsilon)$

$\overset{(7)}{=}$ a $\quad \textsc{FillRec}_1(\to, \to^2, \to)$

$\textsc{FillRec}_1(\downarrow, \to^0, \downarrow)$ $\qquad \# \diamond \mapsto \diamond$



We observe that $\text{lookup}_1 = \{\diamond \mapsto \diamond, (\to {}_2\!\searrow\!\diamond) \mapsto (\to^2\!\downarrow \diamond)\}$ links each position in ζ' that is labeled with a terminal symbol to the corresponding position in ζ. □

By applying Algorithm 5.4.1 to all positions of some $\text{pos}(\zeta)$-decomposition, we obtain the operations of an sDCP algebra. We can construct an sDCP that recognizes ζ leveraging these operations by choosing an alphabet, nonterminals, and rules appropriately.

Definition 5.4.5. Let $\zeta \in \mathrm{U}^*_\Delta$ and t be a $\text{pos}(\zeta)$-decomposition. We construct an sDCP $\mathbb{G}^{\zeta,t} = (G^{\zeta,t}, \mathcal{A}^{\zeta,t})$ with $G^{\zeta,t} = (N, \Sigma^{\zeta,t}, S, R)$ as follows:

- $N = \{t(p) \mid p \in \text{pos}(t)\}$.
- $\Sigma^{\zeta,t} = \{\sigma_p \mid p \in \text{pos}(t)\}$ where, for each $p \in \text{pos}(t)$ with $\text{children}^t(p) = p_1 \cdots p_n$, we let
$$\text{sort}_{\Sigma^{\zeta,t}}(\sigma_p) = (\bar{s}_{t(p_1)} \cdots \bar{s}_{t(p_n)}, \bar{s}_{t(p)}) \ .$$

- $S = \mathrm{pos}(\zeta)$.
- $R = \{t(p) \to \sigma\,(\,t(p_1)\,,\ldots,\,t(p_n)\,) \mid p \in \mathrm{pos}(t), \mathrm{children}^t(p) = p_1 \cdots p_n\}$.
- $\sigma_p^{\mathcal{A}^{\zeta,t}} = f[\zeta, t(p), t(p_1), \ldots, t(p_n)]$
  for each $p \in \mathrm{pos}(t)$ where $\mathrm{children}^t(p) = p_1 \cdots p_n$. □

Example 5.4.6. We consider the hedge ζ and the sets of position U_0, …, U_4 as indicated in Figure 5.5. Because $U_i \supseteq U_{i+1}$ and $U_0 = \mathrm{pos}(\zeta)$, we have that $t = U_0(U_1(U_2(U_3(U_4))))$ is a $\mathrm{pos}(\zeta)$-decomposition. By Definition 5.4.5 we obtain an sDCP $\mathbb{G}^{\zeta,t} = (G^{\zeta,t}, \mathcal{A}^{\zeta,t})$, where the operation of $\mathcal{A}^{\zeta,t}$ are indicated in Figure 5.5 and $G^{\zeta,t} = (N, \Sigma^{\zeta,t}, S, R)$ is as follows:

- $N = \{U_0, \ldots, U_4\}$,
- $\Sigma^{\zeta,t} = \{\sigma_{\downarrow^i\diamond} \mid i \in [4]_0\}$,
- $S = U_0$, and
- $R = \{U_i \to \sigma_{\downarrow^i\diamond}(U_{i+1}) \mid i \in \{0,1,2,3\}\} \cup \{U_4 \to \sigma_{\downarrow^4\diamond}()\}$. □

5.5 Induction of LCFRS/sDCP hybrid grammars

Finally, let us turn to the induction of a hybrid grammar. Let (w, ζ, α) be a hybrid tree and t be a $[|w|]$-decomposition. We first construct a $\mathrm{pos}(\zeta)$-decomposition t' that is isomorphic to t. This construction is different for constituent and dependency trees:

1. Let (w, ζ, α) be a constituent tree. We define the function $\chi\colon \mathrm{pos}(\zeta) \to \mathrm{pos}(\zeta)$ such that

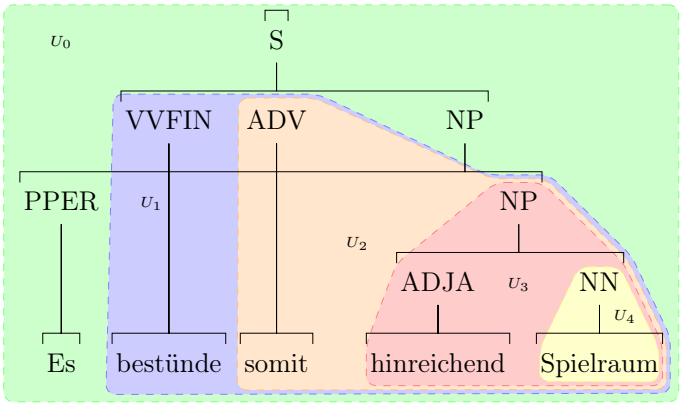
   $$\chi(U) = U \cup \{p \in \mathrm{pos}(\zeta) \mid \mathrm{children}^\zeta(p) \neq \varepsilon \wedge \forall p' \in \mathrm{children}^\zeta(p)\colon p' \in U\}\ .$$

   We construct the $\mathrm{pos}(\zeta)$-decomposition t' where $\mathrm{pos}(t) = \mathrm{pos}(t')$ and, for each $p \in \mathrm{pos}(t)$, we have

   $$t'(p) = \bigcup_{i=0}^{\infty} \chi^i(\{\alpha(i) \mid i \in t(p)\})\ .$$

2. Let (w, ζ, α) be a dependency tree. We construct the $\mathrm{pos}(\zeta)$-decomposition t' where $\mathrm{pos}(t) = \mathrm{pos}(t')$ and, for each $p \in \mathrm{pos}(t)$, we have $t'(p) = \{\alpha(i) \mid i \in t(p)\}$.

Observation 5.5.1. For all $i \in [|w|]$ and $p \in \mathrm{pos}(t)$: $i \in t(p) \setminus \bigcup_{p' \in \mathrm{children}^t(p)} t(p')$ iff $\alpha(i) \in t'(p) \setminus \bigcup_{p' \in \mathrm{children}^{t'}(p)} t'(p')$. □



p	$\mathrm{sort}_{\Sigma^{\zeta,t}}(\sigma_p)$	$\sigma_p^{\mathcal{A}^{\zeta,t}}$
U_0 $\diamond$	$((00),(0))$	$\left\langle \begin{array}{c} \text{S} \\ \hline x_1^1 \quad \text{NP} \\ \hline \text{PPER} \quad\quad x_2^1 \\ \text{Es} \end{array} \right\rangle$
U_1 $\downarrow\diamond$	$((00),(00))$	$\left\langle \begin{array}{c} \text{VVFIN} \quad x_1^1 \\ \text{bestünde} \end{array} , \; x_2^1 \right\rangle$
U_2 $\downarrow^2\diamond$	$((0),(00))$	$\left\langle \begin{array}{c} \text{ADV} \\ \text{somit} \end{array} , \; x_1^1 \right\rangle$
U_3 $\downarrow^3\diamond$	$((0),(0))$	$\left\langle \begin{array}{c} \text{NP} \\ \hline \text{ADJA} \quad\quad x_1^1 \\ \text{hinreichend} \end{array} \right\rangle$
U_4 $\downarrow^4\diamond$	$(\varepsilon,(0))$	$\left\langle \begin{array}{c} \text{NN} \\ \text{Spielraum} \end{array} \right\rangle$

Figure 5.5: A hedge ζ with sets of positions $U_0, \ldots, U_4$ indicated in different colors where $U_i \supset U_{i+1}$. The table lists the operations of the sDCP algebra constructed from the $\mathrm{pos}(\zeta)$-decomposition $t = U_0(U_1(U_2(U_3(U_4))))$.

By Definition 5.2.4 and Definition 5.4.5 we obtain two IRTGs $\mathbb{G}^{w,t} = (G, \mathcal{A}^{w,t})$ and $\mathbb{G}^{\zeta,t'} = (G', \mathcal{A}^{\zeta,t'})$. We observe that the involved grammars are isomorphic because t and t' are isomorphic. We construct the LCFRS/sDCP hybrid grammar $\mathbb{G}^{h,t,t'} = (G, \mathcal{A}^{w,t}, \mathcal{A}^{\zeta,t'}, \mathcal{A}^{h,t,t'})$, where we choose $\mathcal{A}^{h,t,t'}$ to be an alignment algebra such that, for each $p \in \mathrm{pos}(t)$, $\sigma_p^{\mathcal{A}^{h,t,t'}} = \langle u_1, \ldots, u_k; \xi_1, \ldots, \xi_l; \tilde{\alpha} \rangle$, where

- $\langle u_1, \ldots, u_k \rangle = \sigma_p^{\mathcal{A}^{w,t}}$,
- $\langle \xi_1, \ldots, \xi_l \rangle = \sigma_p^{\mathcal{A}^{w,t}}$, and
- $\tilde{\alpha}$ is obtained as follows. Let $i \in t(p) \setminus \bigcup_{p' \in \mathrm{children}^t(p)} t(p')$. Then there are unique numbers $i' \in [k]$ and $j' \in [|u_{i'}|]$ such that $i \sim_p (i', j')$ according to Definition 5.2.2. Let $\alpha(i) = \tilde{p}$ where, by Observation 5.5.1, $\tilde{p} \in t'(p) \setminus \bigcup_{p' \in \mathrm{children}^t(p)} t'(p')$. Let $q \in [l]$ be such that $\tilde{p} \in \mathrm{hspan}^{\zeta}(w_q)$ where $\mathrm{boundary}^{\zeta}(t(p)) = (w_1, \ldots, w_l)$. Then we set $\tilde{\alpha}_q(i', j') = \mathrm{lookup}_q(\tilde{p})$.

Example 5.5.2. Consider the constituent tree (w, ζ, α) where w is as in Example 5.2.6 and ζ is as in Example 5.4.6. Moreover, let t be the $[|w|]$-decomposition from Example 5.2.6. The $\mathrm{pos}(\zeta)$-decomposition t' that is isomorphic to t is exactly the $\mathrm{pos}(\zeta)$-decomposition considered in Example 5.4.6. The alignment function $\tilde{\alpha}$ for each $p \in \mathrm{pos}(t)$ is as follows:

p	$\tilde{\alpha}_1$
$\diamond$	$\{(1,1) \mapsto \downarrow\rightarrow\downarrow\downarrow\diamond\}$
$\downarrow\diamond$	$\{(1,1) \mapsto \downarrow\diamond\}$
$\downarrow^2\diamond$	$\{(1,1) \mapsto \downarrow\diamond\}$
$\downarrow^3\diamond$	$\{(1,1) \mapsto \downarrow\downarrow\diamond\}$
$\downarrow^4\diamond$	$\{(1,1) \mapsto \downarrow\diamond\}$

□

Example 5.5.3. Given a non-projective dependency tree $h = (w, \zeta, \alpha)$, we induce hybrid grammars using different decompositions t of $[|w|]$. In Figure 5.6, we use $t = \{1,2,3,4\}(\{2,3,4\}(\{3,4\}(\{4\})))$ and obtain the corresponding $\mathrm{pos}(\zeta)$-decomposition $t' = \mathrm{pos}(\zeta)(\{\diamond, \downarrow\diamond, \downarrow\rightarrow\diamond\}(\{\downarrow\rightarrow\diamond, \downarrow\diamond\}(\{\downarrow\rightarrow\diamond\})))$. Notably, the constructed LCFRS algebra operates only over 1-tuples. In fact, the operations are of the form $\langle\delta\rangle$ or $\langle\delta\ x_1^1\rangle$, i.e., the LCFRS is a syntactic variant of a Chomsky Type-3 grammar. The sDCP algebra makes use of one component only but uses non-trivial contexts.

With the alternative decomposition $t = \{1,2,3,4\}(\{1,2,3\}(\{1,2\}(\{1\})))$ in Figure 5.7 we obtain again a Chomsky Type-3-restricted LCFRS but the sDCP algebra now employs up to two components. As a last example we consider the decomposition $t = \{1,2,3,4\}(\{1,3,4\}(\{1,3\}(\{3\})))$ in Figure 5.8. It leads to a situation where both the LCFRS and the sDCP use 2 components – however this time the contexts generated by the sDCP are trivial, i.e., they do not contain first-order variables. □

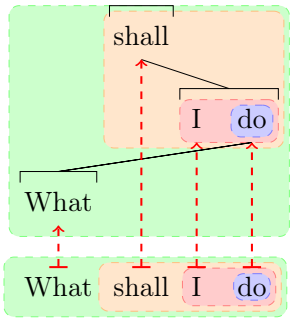
$h = (w, \zeta, \alpha)$	U	p	$\text{sort}_{\Sigma^{\zeta,t'}}(\sigma_p)$	$\sigma_p^{\mathcal{A}^{h,t,t'}}$
	$\{4\}$	$\downarrow^3\diamond$	$(\varepsilon, (1))$	$\left\langle \text{do} \; ; \; \begin{array}{c}\text{do}\\ \lfloor\;\rfloor \\ y_1\end{array} \; ; \; \tilde{\alpha}_1(1,1) = \diamond \right\rangle$
	$\{3,4\}$	$\downarrow^2\diamond$	$((1),(1))$	$\left\langle \text{I } x_1^1 \; ; \; \begin{array}{c}\text{I } x_1^1\\ \lfloor\;\rfloor \\ y_1\end{array} \; ; \; \tilde{\alpha}_1(1,1) = \diamond \right\rangle$
	$\{2,3,4\}$	$\downarrow\diamond$	$((1),(1))$	$\left\langle \text{shall } x_1^1 \; ; \; \begin{array}{c}\text{shall}\\ x_1^1 \\ y_1\end{array} \; ; \; \tilde{\alpha}_1(1,1) = \diamond \right\rangle$
	$\{1,2,3,4\}$	$\diamond$	$((1),(0))$	$\left\langle \text{What } x_1^1 \; ; \; \begin{array}{c}x_1^1\\ \lfloor\;\rfloor \\ \text{What}\end{array} \; ; \; \tilde{\alpha}_1(1,1) = \downarrow\diamond \right\rangle$

Figure 5.6: Induction of a hybrid grammar from a dependency structure – operations of the constructed algebras.

Lexicalized LCFRS. A (Σ, Δ)-LCFRS algebra $\mathcal{A}$ is called *lexicalized LCFRS* if for each $k \in \mathbb{N}$ and $\sigma \in \Sigma_k$ where $\sigma^{\mathcal{A}} = \langle w_1, \ldots, w_k \rangle$, the string $w_1 \cdots w_k$ contains exactly one symbol in Δ. Given a string $w \in \Delta^*$ and a $[|w|]$-decomposition t, we observe that the $(\Sigma^{w,t}, \Delta)$-algebra $\mathcal{A}^{w,t}$ is lexicalized if, for each $p \in \text{pos}(t)$, we have that $|t(p)| = 1 + \sum_{p' \in \text{children}^t(p)} |t(p')|$. Lexicalized LCFRS, i.e., LCFRS that use a lexicalized LCFRS algebra, are of particular interest because they allow for efficient parsing: Each derivation of a sentence w consists of $|w|$ rules – one for each sentence position i that generates $w[i]$. One may further improve parsing speed and accuracy by considering for each sentence position i only a strict subset of the rules that generate $w[i]$. This selection process is also called *supertagging* (Bangalore and Joshi 1999).

We observe that the grammars induced in the previous examples have lexicalized LCFRS algebras. This illustrates that our induction algorithms for LCFRS/sDCP hybrid grammars can be used to construct lexicalized grammars for syntactic parsing to discontinuous constituent trees or non-projective dependency trees as long as a suitable $[|w|]$-decomposition is chosen. Simultaneously, the fanout of the resulting grammar can be controlled. Obviously, this opens a large space of potential strategies to choose $[|w|]$-decompositions that behave favourable in practise. In the experimental

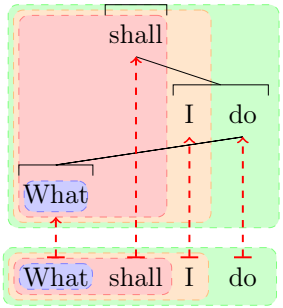
$h = (w, \zeta, \alpha)$	U	p	$\mathrm{sort}_{\Sigma^{\zeta,t'}}(\sigma_p)$	$\sigma_p^{\mathcal{A}^{h,t,t'}}$
	$\{1\}$	$\downarrow^3\diamond$	$(\varepsilon, (0))$	$\left\langle \text{What} \; ; \; \overbrace{\text{What}} \; ; \; \tilde{\alpha}_1(1,1) = \diamond \right\rangle$
	$\{1,2\}$	$\downarrow^2\diamond$	$((0), (1\ 0))$	$\left\langle x_1^1 \text{ shall} \; ; \; \overbrace{\underset{y_1}{\perp}}^{\text{shall}}, \overbrace{x_1^1} \; ; \; \begin{matrix}\tilde{\alpha}_1(1,2) = \diamond \\ \tilde{\alpha}_2 = \emptyset\end{matrix} \right\rangle$
	$\{1,2,3\}$	$\downarrow\diamond$	$((1\ 0), (1\ 0))$	$\left\langle x_1^1 \text{ I} \; ; \; \overbrace{\underset{\text{I } y_1}{\perp}}^{x_1^1}, \overbrace{x_2^1} \; ; \; \begin{matrix}\tilde{\alpha}_1(1,2) = \downarrow\diamond \\ \tilde{\alpha}_2 = \emptyset\end{matrix} \right\rangle$
	$\{1,2,3,4\}$	$\diamond$	$((1\ 0), (0))$	$\left\langle x_1^1 \text{ do} \; ; \; \overbrace{\underset{x_2^1}{\perp}}^{x_1^1} \; ; \; \tilde{\alpha}_1(1,2) = \downarrow\diamond \right\rangle$

Figure 5.7: Induction of a hybrid grammar from a dependency structure – operations of the constructed algebras.

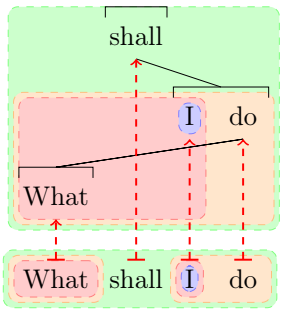
$h = (w, \zeta, \alpha)$	U	p	$\mathrm{sort}_{\Sigma^{\zeta,t'}}(\sigma_p)$	$\sigma_p^{\mathcal{A}^{h,t,t'}}$
	$\{3\}$	$\downarrow^3\diamond$	$(\varepsilon, (0))$	$\left\langle \text{I} \; ; \; \overbrace{\text{I}} \; ; \; \tilde{\alpha}_1(1,1) = \diamond \right\rangle$
	$\{1,3\}$	$\downarrow^2\diamond$	$((0), (0\ 0))$	$\left\langle \text{What}, x_1^1 \; ; \; \overbrace{x_1^1}, \overbrace{\text{What}} \; ; \; \begin{matrix}\tilde{\alpha}_1 = \emptyset \\ \tilde{\alpha}_2(1,1) = \diamond\end{matrix} \right\rangle$
	$\{1,3,4\}$	$\downarrow\diamond$	$((0\ 0), (0))$	$\left\langle x_1^1, x_1^2 \text{ do} \; ; \; \overbrace{\underset{x_2^1}{\perp}}^{x_1^1 \text{ do}} \; ; \; \tilde{\alpha}_1(2,2) = \rightarrow\diamond \right\rangle$
	$\{1,2,3,4\}$	$\diamond$	$((0), (0))$	$\left\langle x_1^1 \text{ shall } x_1^2 \; ; \; \overbrace{\underset{x_1^1}{\perp}}^{\text{shall}} \; ; \; \tilde{\alpha}_1(1,2) = \diamond \right\rangle$

Figure 5.8: Induction of a hybrid grammar from a dependency structure – operations of the constructed algebras.

section, we consider only $[|w|]$-decompositions that are also recursive partitionings (and therefore in general do not lead to lexicalized grammars).

5.6 Induction from a corpus

We described a method for inducing a hybrid grammar from a single hybrid tree (w, ζ, α) that generates exactly this hybrid tree. For a hybrid grammar to be useful, we want it to generate a language of different hybrid trees – on the one hand those found in a certain training set, on the other hand reasonable new ones obtained by recombining building blocks that we obtained from the hybrid trees in the training set. To this end, we construct hybrid grammars for each hybrid tree in our training set (also called training corpus) and merge these hybrid grammars to a single one.

Firstly, we combine the ranked alphabets $\Sigma^{w,t}$ to a single one, such that each pair of symbols σ_1 and σ_2 where $\sigma_1^{\mathcal{A}^{h_1,t_1,t_1'}} = \sigma_2^{\mathcal{A}^{h_2,t_2,t_2'}}$ is merged.

Secondly, the nonterminals are renamed as follows: Let $h = (w, \zeta, \alpha)$ be a hybrid tree and $\mathbb{G}^{h,t,t'}$ be the hybrid grammar constructed for h and decompositions t and t'. Let $p \in \mathrm{pos}(t)$ and $U = t(p)$. We rename the nonterminal U according to one of different *nonterminal labeling strategies*.

Let $U' = t'(p)$ and $\mathrm{boundary}^{\zeta}(U') = (w_1, \ldots, w_l)$. A nonterminal $(w_1'; \cdots; w_l'; ; k)$ is constructed as follows:

- For each $i \in [l]$, let $w_i = ((p_1, \to^{j_1}) \cdots (p_{s_i}, \to^{j_{s_i}}), (p_0, \to^{j_0}))$. Then we construct $w_i' = ((v_1; \ldots; v_{s_i}); v_0)$, where, for each $m \in [s_i]_0$, we set
$$v_m = \zeta(p_m\diamond)\ \zeta(p_m \to \diamond)\ \cdots\ \zeta(p_m \to^{j_m - 1} \diamond)\ .$$
- k is the fanout of U.

In this way, we obtain what is called *strict labeling* in Gebhardt, Nederhof, and Vogler (2017).

Alternatively, a sequence v_m may be collapsed, if $j_m > 1$ and $p_m\diamond$ has a parent. In this case, v_m is set to *children-of*$\langle\zeta(\mathrm{parent}(p_m\diamond)\rangle$ where *children-of* is some fresh symbol. This labeling strategy is called *child labeling* in Gebhardt, Nederhof, and Vogler (2017). Usually we expect child labeling to result in a smaller overall number of nonterminals than strict labeling.

Lastly, because Gebhardt (2018) found that initial nonterminal granularity matters for split/merge refinement, we will investigate if a form of Markovization (Johnson 1998; Klein and Manning 2003) applied to strict labeling is beneficial. Precisely, the string v_m is truncated to
$$\zeta(p_m \to^0 \diamond)\ \cdots\ \zeta(p_m \to^{\bar{\mathrm{h}}} \diamond)\ \mathit{trunc}$$
if j_m is larger than a particular value $\bar{\mathrm{h}} \in \mathbb{N}$ where *trunc* is a fresh symbol. Moreover, we may add vertical context by prepending the labels of up to $\bar{\mathrm{v}}$ predecessors to

v_m, if they exist (i.e., $\zeta(\text{parent}(p_m\diamond))$ $\zeta(\text{parent}(\text{parent}(p_m\diamond))$ $\cdots$ $\zeta(\text{parent}^{\bar{\text{v}}}(p_m\diamond))$). We denote the resulting nonterminal labeling strategy strict-Markov-v-$\bar{\text{v}}$-h-$\bar{\text{h}}$ in the following.

Further refinements Very often, the labels of ζ can be decomposed. For instance, during dependency parsing, the label may be composed of a surface word form, a lemma, a part-of-speech tag, morphological information, and a dependency relation. Likely, using all this information leads to scarce data problems. Hence, we will often project from the full label to a selection of these fields, e.g., just the dependency relations, just the part-of-speech tag or, both of the previous ones.

6 Charts of hybrid grammars

For training and decoding it is necessary to compute compact representations of the set of all derivations of a grammar $\mathbb{G}$ for a particular domain object a at consideration. These compact representation are called *parse charts* or just *charts* (Kay 1996), *intersection grammars* (Nederhof and Satta 2003), or, because a derivation is formally a tree and the derivations are represented such that shared subderivations are stored only once, also *packed* or *shared forest* (Lang 1994). In this thesis, we formalize such a representation as a particular regular tree grammar G_a. Equivalently, they could be viewed as hypergraphs (for an overview cf. L. Huang 2008a) or tree automata.

In this section, we give constructions that show how such a packed representation of the set of derivation trees can be computed for LCFRS, sDCP, and hybrid grammars. All these constructions are inspired by a construction by Bar-Hillel, Perles, and Shamir (1961), who showed that context-free string languages are closed under intersection with regular string languages. The constructions we provide differ in two aspects. Instead of intersecting the language generated by the grammar formalism at hand with an arbitrary regular string, hedge, or hybrid tree language, respectively, we just consider the special case of the regular language that contains only a single string, hedge, or hybrid tree.[1] Note that at least for LCFRS and sDCP also the unrestricted result can be generalized. For LCFRS the closure under intersection with regular languages has been proved by Seki et al. (1991, Theorem 3.9). To obtain a similar result for the variant of sDCP we use, one would need to define or select a suitable automata model for unranked hedges first.

The second difference is that the involved grammars (LCFRS/sDCP) are more expressive (multiple components, second-order substitution) than context-free grammars. This effects the notational effort needed in proofs and construction to a point where statements that are straightforward for CFG are not as easy to see but simultaneously cumbersome to proof in an axiomatic fashion. Therefore, we restrict ourselves to only providing proof sketches for theorems and auxiliary lemmas in certain cases and omit detailed, axiomatic proofs.

A construction for the special case of intersection just a single sentence with an LCFRS was first given by Seki et al. (1991). Precisely, they give a parsing algorithm

[1] Because of this underlying intersection of a context-free and a regular device, this approach is also called *parsing as intersection* (Nederhof and Satta 2003; Grune and Jacobs 2008).

that solves the recognition problem for multiple context-free grammars. We are not aware that a similar direct construction was already devised for sDCP and hybrid grammars. In Drewes, Gebhardt, and Vogler (2016) we gave a more general construction for aligned hyperedge replacement bimorphisms – a particular form of synchronized graph grammars. There, we also illustrated that an LCFRS/sDCP hybrid grammar can be encoded as such an aligned hyperedge replacement bimorphism. Although more general, hyperedge replacement grammars tend to have a great expressiveness and, in general, their parsing problem is NP-complete (see Lautemann 1990, for a discussion). Moreover, in the process of hyperedge replacement, one distinguishes abstract and concrete graphs and always ensures that fresh disjoint copies of graphs are used. On the one hand, this abstracts away from the technical position tracing we employ in this thesis for the synchronization of string and tree positions. On the other hand, it leaves the implementation rather vague. In particular, ensuring disjointness of the involved subgraphs during a hyperedge replacement yields a considerable overhead at runtime. We give a new, direct construction for intersecting hedges with sDCP and hybrid trees with LCFRS/sDCP hybrid grammars.

6.1 Charts for LCFRS

We recall a chart construction for LCFRS. It is conceptionally similar to the deductive parsing algorithm presented, for instance, by Kallmeyer (2010) but adjusted to yield an RTG. However, we follow Koller and Kuhlmann (2011) by splitting the construction in two steps: first we construct a decomposition (i.e., an RTG) of a string w in a particular LCFRS algebra. Afterward, this decomposition is intersected with the RTG that is part of the given LCFRS. In particular, each nonterminal of the decomposition is a tuple of disjoint intervals of w. A rule $A_0 \to \sigma(A_1, \ldots, A_n)$ is added to the decomposition, if the interpretation of σ in the LCFRS algebra combines the substrings in the intervals specified by A_1, …, A_n to the substrings specified by A_0. As initial nonterminal symbol we choose the tuple with a single interval that covers the entire sentence. Example 6.1.4 illustrates this construction.

Theorem 6.1.1. Let Σ be a ranked alphabet, Δ be an alphabet, and $\mathcal{A}$ be a (Σ, Δ)-LCFRS algebra. $\mathcal{A}$ is regular decomposable. □

Proof. Let $w \in \Delta^*$ with $|w| = \ell$. Construct the RTG $G = (N, \Sigma, S, R)$ (called *$\mathcal{A}$-decomposition of w*) with

- $N = \{\langle (i_1, j_1), \ldots, (i_k, j_k)\rangle \mid k \in [k_{\max}], 0 \leq i_1 \leq j_1 \leq \ell, \ldots, 0 \leq i_k \leq j_k \leq \ell,$ $\forall \kappa, \kappa' \in [k]\colon \kappa \neq \kappa'$ implies $[i_\kappa + 1, j_\kappa] \cap [i_{\kappa'} + 1, j_{\kappa'}] = \emptyset\}$
- $S = \langle (0, \ell)\rangle$
- R contains each rule of the form

$$A_0 \to \sigma(A_1, \ldots, A_n)$$

where

- $\sigma \in \Sigma_{(k_1 \cdots k_n, k_0)}$ for $k_0, \ldots, k_n \in \mathbb{N}$ and $n \in \mathbb{N}$,
- for each $q \in [n]_0$
$$A_q = \langle (i_1^q, j_1^q), \ldots, (i_{k_q}^q, j_{k_q}^q) \rangle$$
with $0 \leq i_p^q \leq j_p^q \leq \ell$ for each $p \in [k_q]_0$, and
- $w[A_0] = \sigma^{\mathcal{A}}(w[A_1], \ldots, w[A_n])$ where we denote
$$w[A_i] = (w[i_1^q : i_1^q], \ldots, w[i_{k_q}^q : j_{k_q}^q]) \ .$$

The following two lemmas imply soundness and completeness of the construction and are easy to prove by structural induction on d and t, respectively. To show completeness one also leverages the linearity and non-deleting property of the LCFRS operations.

Lemma 6.1.2. Let $A \in N$ and $d \in (\mathrm{T}_R)_A$. Then $\left[\!\left[[\![d]\!]_A^{\Pi} \right]\!\right]_k^{\mathcal{A}} = w[A]$. □

Lemma 6.1.3. Let $t \in (\mathrm{T}_\Sigma)_k$ and $A \in N$ be such that $[\![t]\!]_k^{\mathcal{A}} = w[A]$. Then there exists $d \in (\mathrm{T}_R)_A$ with $[\![d]\!]_A^{\Pi} = t$. □

Finally we proof $L(G) = \{t \in (\mathrm{T}_\Sigma)_1 \mid [\![t]\!]_1^{\mathcal{A}} = (w)\}$.

- $\subseteq$ Let $t \in L(G)$. Then $\exists d \in (\mathrm{T}_R)_S$ such that $[\![d]\!]_S^{\Pi} = t$. By Lemma 6.1.2 we have $[\![t]\!]_1^{\mathcal{A}} = w[S] = (w[0 : \ell]) = (w)$.
- $\supseteq$ Let $t \in (\mathrm{T}_\Sigma)_1$ such that $[\![t]\!]_1^{\mathcal{A}} = (w)$. For S we have $[\![t]\!]_1^{\mathcal{A}} = w[S]$. By Lemma 6.1.3 there is $d \in (\mathrm{T}_R)_S$ with $[\![d]\!]_S^{\Pi} = t$. Thus $t \in L(G)$. ■

We illustrate the construction of the LCFRS decomposition in the following example.

Example 6.1.4 (Example 4.4.2 cont'd)**.** Let $\Delta = \{a, b, c\}$ be an alphabet, let $(\Sigma, \mathrm{sort}_{\mathrm{LCFRS}})$ be an $\mathbb{N}$-sorted alphabet, and let $\mathcal{A}$ be a (Σ, Δ)-LCFRS algebra where $\Sigma = \{\sigma, \gamma, \delta, \beta\}$ and

$$\begin{aligned}
\mathrm{sort}_{\mathrm{LCFRS}}(\sigma) &= (2\ 1, 1) & \sigma^{\mathcal{A}} &= \langle x_1^2 x_1^1 x_2^1 \rangle \\
\mathrm{sort}_{\mathrm{LCFRS}}(\gamma) &= (2\ 1, 1) & \gamma^{\mathcal{A}} &= \langle x_1^1 a, a x_2^1 \rangle \\
\mathrm{sort}_{\mathrm{LCFRS}}(\delta) &= (\varepsilon, 2) & \delta^{\mathcal{A}} &= \langle c, c \rangle \\
\mathrm{sort}_{\mathrm{LCFRS}}(\beta) &= (\varepsilon, 1) & \beta^{\mathcal{A}} &= \langle b \rangle \ .
\end{aligned}$$

Consider the string $w = {}_0b_1c_2a_3a_4a_5a_6c_7$ (with string-range positions added for clarity). The $\mathcal{A}$-decomposition of w has $\langle (0, 7) \rangle$ as initial nonterminal and the following rules:

- for each $i, k, l, j \in [7]_0$ with $0 \leq i \leq k \leq l \leq j \leq 7$:

$$\langle (i,j) \rangle \to \sigma(\ \langle (k,l),(l,j) \rangle\ ,\ \langle (i,k) \rangle\ )$$

- for each $i_1, i_2, j_1, j_2 \in [7]_0$ with $[i_1+1, j_1] \cap [i_2+1, j_2] = \emptyset$, $j_1 \in \{3,4,5,6\}$, and $i_2 \in \{2,3,4,5\}$:

$$\langle (i_1,j_1),(i_2,j_2) \rangle \to \gamma(\ \langle (i_1, j_1-1),(i_2+1,j_2) \rangle\ )$$

- $\langle (1,2),(6,7) \rangle \to \delta(\ )$ and $\langle (6,7),(1,2) \rangle \to \delta(\ )$, and
- $\langle (0,1) \rangle \to \beta(\ )$.

Apparently, there is only one derivation tree in $(\mathrm{T}_R)_{\langle (0,7) \rangle}$:

$$\begin{array}{c} \langle (0,7) \rangle \to \sigma(\ \langle (1,4),(4,7) \rangle\ ,\ \langle (0,1) \rangle\ ) \\ \swarrow \qquad\qquad \searrow \\ \langle (1,4),(4,7) \rangle \to \gamma(\ \langle (1,3),(5,7) \rangle\ ) \qquad \langle (0,1) \rangle \to \beta(\ ) \\ | \\ \langle (1,3),(5,7) \rangle \to \gamma(\ \langle (1,2),(6,7) \rangle\ ) \\ | \\ \langle (1,2),(6,7) \rangle \to \delta(\ ) \end{array}$$

□

The corollary follows directly from the previous theorem and the fact that RTG are closed under intersection.

Corollary 6.1.5. Let $\mathbb{G} = (G, \mathcal{A})$ be an LCFRS and $w \in \Delta^*$. Then there is an RTG G_w with $L(G_w) = \{t \in L(G) \mid [\![t]\!]_1^{\mathcal{A}} = (w)\}$. □

In particular, let $D^{\mathcal{A}}(w)$ be the $\mathcal{A}$-decomposition of w. Using the product construction from Definition 2.3.17 we construct $G_w = G \times D^{\mathcal{A}}(w)$. In order to apply the training framework from Chapter 3, we define the grammar morphism φ_w^G from G_w to G by setting $\varphi_w^G((B,C)) = B$. Recall that B is a nonterminal of G and C is a nonterminal from the $\mathcal{A}$-decomposition of w.

Complexity. To analyze the parsing complexity of constructing the reduct for LCFRS, we just consider the size of the decomposition $D^{\mathcal{A}}(w) = (N, \Sigma, S, R)$. Obviously $|N| \leq (\ell+1)^{2 \cdot k_{\max}}$, because each nonterminal consists of at most $k_{\max}$ pairs of integers in $[\ell]_0$. The number of rules can therefore be bounded by $|R| \leq |\Sigma| \cdot |N|^{n_{\max}+1}$. To give a tighter bound on R, one can consider the *degree* (Seki et al. 1991) of some rule $A_0 \to \sigma(A_1, \ldots, A_n)$. For instance, the number of rules with the terminal σ where $\sigma^{\mathcal{A}} = \langle x_1^1 x_1^2, x_1^3 x_2^1, x_2^2 x_2^3 \rangle$ is strictly smaller than

$$(\ell+1)^{2 \cdot 3} \cdot (\ell+1)^{2 \cdot 2} \cdot (\ell+1)^{2 \cdot 2} \cdot (\ell+1)^{2 \cdot 2} = (\ell+1)^{18}$$

because the end of the interval substituted for x_1^1 determines the beginning of the interval substituted for x_1^2, etc. In fact there are just 9 positions of the string that can be chosen freely[2]. In this case, 9 equals $3 + 2 + 2 + 2$, which is the degree of the rule defined as the sum of the fanouts of the nonterminal occurring in it. In general, one can show that

$$|R| \leq |\Sigma| \cdot (\ell + 1)^{(n_{\max}+1) \cdot k_{\max}} \quad .$$

If we assume that checking $w[A_0] = \sigma^{\mathcal{A}}(w[A_1], \ldots, w[A_n])$ can be done in constant time, then the upper bound on R is also an upper bound on the asymptotic time required to construct $D^{\mathcal{A}}(w)$. The size of G_w is bounded by the product of the size of G and $D^{\mathcal{A}}(w)$.

Due to the strong influence of the parameters $n_{\max}$ and $k_{\max}$, methods for reducing the rank and the fanout have been researched. Notably, the class of LCFRS with $k_{\max} = \kappa + 1$ generates a strict superset of the class of LCFRS with $k_{\max} = \kappa$ (Seki et al. 1991). However, it is possible to find an equivalent binarized LCFRS, i.e., one where $n_{\max} = 2$, for every LCFRS (Seki et al. 1991), but in general this requires an increase in fanout. Hence, Gómez-Rodríguez et al. (2009) consider the problem to give an equivalent binary LCFRS such that the fanout is minimized. Gildea (2010) considers the problem of constructing an equivalent LCFRS where fanout and rank are balanced such that the theoretic parsing complexity is optimal. See also Section 8.5 for further discussion.

6.2 Charts for sDCP

Next we give a chart construction for the variant of sDCP that is used in this thesis. Again, we first show regular decomposability of an arbitrary sDCP algebra. Afterward, the reduct is obtained by intersecting the RTG in an sDCP with the decomposition grammar.

Theorem 6.2.1. Let Σ be a ranked alphabet, Δ be an alphabet, and $\mathcal{A}$ be an (Σ, Δ)-sDCP algebra. $\mathcal{A}$ is regular decomposable. □

Proof of Theorem 6.2.1. We start by presenting and illustrating the construction of the sDCP decomposition for some hedge ζ before we prove its correctness. The construction resembles that for LCFRS in that every nonterminal needs to correspond to a tuple of non-overlapping parts of ζ, but now these non-overlapping parts are hedge contexts. They need to be non-overlapping because the operations of an sDCP algebra are linear and non-deleting. Formally, we utilize the concept of stencil boundary introduced in Definition 5.3.4, i.e., each nonterminal is a tuple of non-overlapping stencil boundaries for ζ. We add each rule $B_0 \to \sigma(B_1, \ldots, B_n)$ where $\sigma^{\mathcal{A}}$ rewrites

[2] Precisely, there are still ordering constraints, which we ignore, e.g., that the beginning of each interval needs to be smaller than its end.

the hedge contexts described by $B_1, \ldots, B_n$ to those described by B_0. We choose as initial symbol the tuple with a single stencil boundary that covers the entire hedge ζ without a gap.

Construction 6.2.2. Let $\zeta \in \mathrm{U}^*_\Delta$. Construct an RTG $G = (N, \Sigma, S, R)$ as follows:

- Let $k \in \mathbb{N}$ and $s_1, \ldots, s_k \in \mathbb{N}$ be such that $(\mathrm{T}_\Sigma)_{s_1 \cdots s_k} \neq \emptyset$. Let $w_1 \in \mathrm{SB}^{\ltimes}_{s_1}(\zeta)$, $\ldots$, $w_k \in \mathrm{SB}^{\ltimes}_{s_k}(\zeta)$ where, for each $o \in [k]$, we let

  $$w_o = ((w_o^1, \to^{i_o^1}) \cdots (w_o^{s_o}, \to^{i_o^{s_o}}), (w_o^0, \to^{i_o^0})) \ .$$

  If, for each $i, j \in [k]$ with $i < j$, we have that w_i and w_j are non-overlapping, then $\langle w_1, \ldots, w_k \rangle \in N_{s_1 \cdots s_k}$. We define, for each $o \in [k]$,

  $$\zeta[w_o] = \zeta|^{\to^{i_o^0}}_{w_o^0} \ltimes \{(w_o^{o'}, \to^{i_o^{o'}}) \mapsto y_{o'} \mid o' \in [s_o]\})$$

  and

  $$\zeta[\langle w_1, \ldots, w_k \rangle] = (\zeta[w_1], \ldots, \zeta[w_k]) \ .$$

- $S = \langle (\varepsilon, (\varepsilon, \to^{\mathrm{len}(\zeta)})) \rangle$.

- $R = \{B_0 \to \sigma(B_1, \ldots, B_n) \mid n \in \mathbb{N}, u_0, \ldots, u_n \in \mathbb{N}^*,$
  $\sigma \in \Sigma_{u_1 \cdots u_n, u_0}, B_0 \in N_{u_0}, \ldots, B_n \in N_{u_n},$
  $\zeta[B_0] = \sigma^{\mathcal{A}}(\zeta[B_1], \ldots, \zeta[B_n])\}$. □

Example 6.2.3 (Example 4.4.2 cont'd)**.** Let $\Delta = \{a, b, c\}$ be an alphabet, let $(\Sigma, \mathrm{sort}_{\mathrm{sDCP}})$ be an $\mathbb{N}^* \times \mathbb{N}$-sorted alphabet, and let $\mathcal{A}$ be a (Σ, Δ)-sDCP algebra where $\Sigma = \{\sigma, \gamma, \delta, \beta\}$ and

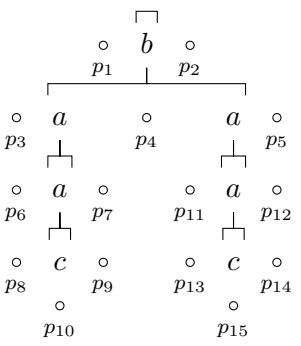
$$\mathrm{sort}_{\mathrm{sDCP}}(\sigma) = ((0\ 0)\ (1), (0)) \qquad \sigma^{\mathcal{A}} = \langle \begin{array}{c} x_1^2 \\ \hline x_1^1 x_2^1 \end{array} \rangle$$

$$\mathrm{sort}_{\mathrm{sDCP}}(\gamma) = ((0\ 0), (0\ 0)) \qquad \gamma^{\mathcal{A}} = \langle \begin{array}{c} a \\ \hline x_1^1 \end{array} , \begin{array}{c} a \\ \hline x_2^1 \end{array} \rangle$$

$$\mathrm{sort}_{\mathrm{sDCP}}(\delta) = (\varepsilon, (0\ 0)) \qquad \delta^{\mathcal{A}} = \langle \ c \ , \ c \ \rangle$$

$$\mathrm{sort}_{\mathrm{sDCP}}(\beta) = (\varepsilon; (1)) \qquad \beta^{\mathcal{A}} = \langle \begin{array}{c} b \\ \hline y_1 \end{array} \rangle \ .$$

Let ζ be as in Figure 6.1. The $\mathcal{A}$-decomposition $G = (N, \Sigma, S, R)$ of ζ has the initial nonterminal $S = \langle (\varepsilon, (\varepsilon, \to)) \rangle$ and R contains the following rules:



$p_1 = \varepsilon$ $\quad p_8 = \downarrow^3$

$p_2 = \rightarrow$ $\quad p_9 = \downarrow^3\rightarrow$

$p_3 = \downarrow$ $\quad p_{10} = \downarrow^4$

$p_4 = \downarrow\rightarrow$ $\quad p_{11} = \downarrow\rightarrow\downarrow$

$p_5 = \downarrow\rightarrow^2$ $\quad p_{12} = \downarrow\rightarrow\downarrow\rightarrow$

$p_6 = \downarrow^2$ $\quad p_{13} = \downarrow\rightarrow\downarrow^2$

$p_7 = \downarrow^2\rightarrow$ $\quad p_{14} = \downarrow\rightarrow\downarrow^2\rightarrow$

$p_{15} = \downarrow\rightarrow\downarrow^3$

Figure 6.1: A hedge ζ and its span positions.

- for each $i \in [15]$, $i_o, j_1, j_2 \in \mathbb{N}$, and $p \in \{\rightarrow, \downarrow\}^*$ such that $(p_i, \rightarrow^{i_o})$ is a span position pair for ξ and $((p, \rightarrow^{j_1+j_2}), (p_i, \rightarrow^{i_o})) \in \mathrm{SB}_1^{\ltimes}(\xi)$:

$$\langle(\varepsilon, (p_i, \rightarrow^{i_o}))\rangle \rightarrow \sigma(\ \langle(\varepsilon, (p_i p, \rightarrow^{j_1})), (\varepsilon, (p_i p \rightarrow^{j_1}, \rightarrow^{j_2}))\rangle\ ,\ \langle((p, \rightarrow^{j_1+j_2}), (p_i, \rightarrow^{i_o}))\rangle\ )$$

- for each $(i, j) \in \{(6, 11), (11, 6), (3, 11), (11, 3), (6, 4), (4, 6), (3, 4), (4, 3)\}$:

$$\langle(\varepsilon, (p_i, \rightarrow)), (\varepsilon, (p_j, \rightarrow))\rangle \rightarrow \gamma(\ \langle(\varepsilon, (p_i \downarrow, \rightarrow))\rangle\ )$$

- $\langle((\downarrow, \rightarrow^2), (p_1, \rightarrow))\rangle \rightarrow \beta(\ )$
- $\langle(\varepsilon, (p_8, \rightarrow)), (\varepsilon, (p_{13}, \rightarrow))\rangle \rightarrow \delta(\ )$ and $\langle(\varepsilon, (p_{13}, \rightarrow)), (\varepsilon, (p_8, \rightarrow))\rangle \rightarrow \delta(\ )$

The only derivation tree in $(\mathrm{T}_R)_S$ is

$$\begin{array}{c}
\langle(\varepsilon, (\varepsilon, \rightarrow))\rangle \rightarrow \sigma(\ \langle(\varepsilon, (p_3, \rightarrow)), (\varepsilon, (p_4, \rightarrow))\rangle\ ,\ \langle((\downarrow, \rightarrow^2), (p_1, \rightarrow))\rangle\ ) \\
\langle(\varepsilon, (p_3, \rightarrow)), (\varepsilon, (p_4, \rightarrow))\rangle \rightarrow \gamma(\ \langle(\varepsilon, (p_6, \rightarrow)), (\varepsilon, (p_{11}, \rightarrow))\rangle\ ) \qquad \langle((\downarrow, \rightarrow^2), (p_1, \rightarrow))\rangle \rightarrow \beta(\ ) \\
\langle(\varepsilon, (p_6, \rightarrow)), (\varepsilon, (p_{11}, \rightarrow))\rangle \rightarrow \gamma(\ \langle(\varepsilon, (p_8, \rightarrow)), (\varepsilon, (p_{13}, \rightarrow))\rangle\ ) \\
\langle(\varepsilon, (p_8, \rightarrow)), (\varepsilon, (p_{13}, \rightarrow))\rangle \rightarrow \delta(\ )
\end{array}$$

□

Lemma 6.2.4. Let $\zeta \in \mathrm{U}_{\Delta}^*$, $\mathcal{A}$ be a (Σ, Δ)-sDCP-algebra, and $G = (N, \Sigma, S, R)$ be the $\mathcal{A}$-decomposition of ζ. For each $u_0 \in \mathbb{N}^*$, $B \in N_{u_0}$, and $d \in (\mathrm{T}_R)_B$ it holds that $\left[\!\left[[\![d]\!]_B^{\Pi} \right]\!\right]_{u_0}^{\mathcal{A}} = \zeta[B]$. □

Proof. By structural induction on d. Let $d = (B \to \sigma(B_1, \dots, B_n))(d_1, \dots, d_n)$ with $n \in \mathbb{N}$, $u_1, \dots, u_n \in \mathbb{N}^*$, $\sigma \in \Sigma_{u_1 \cdots u_n, u_0}$, $B_1 \in N_{u_1}$, $\dots$, $B_n \in N_{u_n}$, and $d_1 \in (\mathrm{T}_R)_{B_1}, \dots, d_n \in (\mathrm{T}_R)_{B_n}$. Then

$$\begin{aligned}
&\left[\!\left[[\![(B \to \sigma(B_1, \dots, B_n))(d_1, \dots, d_n)]\!]_B^{\Pi} \right]\!\right]_{u_0}^{\mathcal{A}} \\
&= \left[\!\left[\sigma([\![d_1]\!]_{B_1}^{\Pi}, \cdots, [\![d_n]\!]_{B_n}^{\Pi}) \right]\!\right]_{u_0}^{\mathcal{A}} \\
&= \sigma^{\mathcal{A}}(\left[\!\left[[\![d_1]\!]_{B_1}^{\Pi} \right]\!\right]_{u_1}^{\mathcal{A}}, \cdots, \left[\!\left[[\![d_n]\!]_{B_n}^{\Pi} \right]\!\right]_{u_n}^{\mathcal{A}}) \\
&= \sigma^{\mathcal{A}}(\zeta[B_1], \dots, \zeta[B_n]) && \text{(induction hypothesis)} \\
&= \zeta[B] && \text{(by construction)}
\end{aligned}$$

■

Lemma 6.2.5. Let $\zeta \in \mathrm{U}_{\Delta}^*$, $\mathcal{A}$ be a (Σ, Δ)-sDCP-algebra, and $G = (N, \Sigma, S, R)$ be the $\mathcal{A}$-decomposition of ζ. Let $u_0 \in \mathbb{N}^*$, $B \in N_{u_0}$, and $t \in (\mathrm{T}_\Sigma)_{u_0}$. Then $[\![t]\!]_{u_0}^{\mathcal{A}} = \zeta[B]$ implies that there is $d \in (\mathrm{T}_R)_B$ such that $[\![d]\!]_B^{\Pi} = t$. □

Proof. By structural induction on t. Let $t = \sigma(t_1, \dots, t_n)$ for $n \in \mathbb{N}$, $u_1, \dots, u_n \in \mathbb{N}^*$, $\sigma \in \Sigma_{(u_1 \cdots u_n, u_0)}$, $t_1 \in (\mathrm{T}_\Sigma)_{u_1}$, $\dots$, $t_n \in (\mathrm{T}_\Sigma)_{u_n}$, and $B = \langle w_1^0, \dots, w_{|u_0|}^0 \rangle$. Then $\zeta[B] = [\![t]\!]_{u_0} = \sigma^{\mathcal{A}}([\![t_1]\!]_{u_1}^{\mathcal{A}}, \dots, [\![t_n]\!]_{u_n}^{\mathcal{A}})$. Let $k = |u_0|$ and $\xi_1, \dots, \xi_k$ be such that $\sigma^{\mathcal{A}} = \langle \xi_1, \dots, \xi_k \rangle$. Because $\xi_1, \dots, \xi_k$ contain every second-order variable $x_1^1, \dots, x_{|u_1|}^1$, $\dots$, $x_1^n, \dots, x_{|u_n|}^n$ exactly once (together)[3], there are pairwise non-overlapping stencil boundaries $w_1^1, \dots, w_{|u_1|}^1, \dots, w_1^n, \dots, w_{|u_n|}^n$ where, for each $i \in [n]$ and $j \in [|u_i|]$, there exists $j' \in [|u_0|]$ such that w_j^i is encompassed by $w_{j'}^0$ and where, for each $i \in [n]$, $\zeta[\langle w_1^i, \dots, w_{|u_i|}^i \rangle] = [\![t_i]\!]_{u_i}^{\mathcal{A}}$. For each $i \in [n]$, let $B_i = \langle w_1^i, \dots, w_{|u_i|}^i \rangle$. Then, for each $i \in [n]$, we obtain by induction that there is a derivation $d_i \in (\mathrm{T}_R)_{B_i}$ with $[\![d_i]\!]_{B_i}^{\Pi} = t_i$. By construction there is a rule $\varrho = B \to \sigma(B_1, \dots, B_n)$. Hence, $[\![\varrho(d_1, \dots, d_n)]\!]_B^{\Pi} = t$. ■

Theorem 6.2.1 directly follows from Lemma 6.2.4 and Lemma 6.2.5 by applying them to the initial nonterminal S and the observation that $\zeta[S] = (\zeta)$.

Complexity. Similar as for the decomposition for LCFRS algebras, we derive upper bounds on the number of nonterminals and rules of the decomposition of an sDCP algebra for some hedge ζ. Let $G = (N, \Sigma, S, R)$ be as in Construction 6.2.2 and $|\mathrm{spos}(\zeta)| = p$. Each nonterminal $B \in N$ is of the form $\langle w^1, \dots, w^l \rangle$, where each w^j has the form $((w_1^j, \to^{i_1^j}) \cdots (w_{s_j}^j, \to^{i_{s_j}^j}), (w_0^j, \to^{s_0^j}))$. Hence, we have at most p^2

[3] I.e., $\langle \xi_1, \dots, \xi_k \rangle$ is linear and non-deleting.

options to choose each $(w_q^j, \rightarrow^{i_q^j})$ and, thus, at most $p^{2(s_j+1)}$ options to choose w^j. To uniquely determine B, we need to choose no more than $p^{(2\cdot(s_{\max}+1)\cdot l_{\max})}$ span positions, where $s_{\max}$ and $l_{\max}$ denote the maximum values for s_j and l such that $(\mathrm{T}_\Sigma)_{(s_1,\ldots,s_l)} \neq \emptyset$, respectively. Thus:

$$|N| \leq p^{(2\cdot(s_{\max}+1)\cdot l_{\max})} \quad .$$

For a first upper bound on the numbers of rules, we can then simply choose

$$|R| \leq |\Sigma| \cdot |N|^{n_{\max}+1} \quad ,$$

where $n_{\max}$ is the maximum value n such that $\Sigma_{(w^1,\ldots,w^n)} \neq \emptyset$. In order to make this bound tighter, we observe that the condition

$$\zeta[B_0] = \sigma^{\mathcal{A}}(\zeta[B_1], \ldots, \zeta[B_n]) \qquad (*)$$

implies dependencies between the span positions chosen for the nonterminals B_0, …, B_n. We observe that for $\sigma^{\mathcal{A}} = \langle \xi_1, \ldots, \xi_{l_0} \rangle$ it suffices to know the start of each span for each first and second order variable occurring in any of ξ_i and the end of each span for each of $\xi_1, \ldots, \xi_{l_0}$.[4] Hence, we can refine the bound on R as follows:

$$\begin{aligned} |R| &\leq \sum_{\bar{s}=((s_1^1 \cdots s_{l_1}^1)\cdots(s_1^n \cdots s_{l_n}^n),(s_1^0 \cdots s_{l_0}^0))} |\Sigma_{\bar{s}}| \cdot p^{(\sum_{i=0}^{n}\sum_{j=1}^{l_i} s_j^i + 1)} \\ &\leq |\Sigma| \cdot p^{(n_{\max}+1)\cdot l_{\max}\cdot(s_{\max}+1)} \quad . \end{aligned}$$

Assuming that the condition $(*)$ can be checked in linear time in $\sum_{i\in[k]} |\mathrm{pos}(\xi_i)|$ where $\sigma^{\mathcal{A}} = \langle \xi_1, \ldots, \xi_k \rangle$, the asymptotic time required to compute G is limited by $|R|$.

6.3 Charts for LCFRS/sDCP hybrid grammars

We conclude this section by providing a construction for the chart of an LCFRS/sDCP hybrid grammar. We construct a joint decomposition for the LCFRS algebra, the

[4] Let x_r^q be a second-order variable occurring in ξ_j. Then the end position spanned by it can be derived as follows:

- If there occurs another variable v behind x_r^q, then we know v's start position, and we can easily calculate the end of ξ_r^q.
- Otherwise, if x_r^q has no parent in ξ_j, then we know the end position of ξ_j, and we easily calculate the end of ξ_r^q.
- Otherwise, if x_r^q is the m-th child of some node n in ξ_j, then we know n's position in ζ and the end position of n's m-th child. Again we can simply calculate the end position of x_r^q.

A similar argument can be made for first-order variables. Likewise, the beginning of the span for ξ_j can be calculated from the distance to the end or ξ_j or the first variable occurring in ξ_j.

sDCP algebra, and for the alignment algebra first. Then we intersect the RTG of the hybrid grammar with the decomposition.

Let $(\bar{w}, \bar{\zeta}, \bar{\alpha})$ be a hybrid tree. The construction employs nonterminals of the form $(A, B, \tilde{\alpha})$ where A is a nonterminal of the LCFRS decomposition (see item (a)), B is a nonterminal of the sDCP decomposition (see item (b)), and $\tilde{\alpha}$ is a function that maps the string positions covered by A to hedge positions covered by B. Only if this function matches $\bar{\alpha}$ on the positions in A, $(A, B, \tilde{\alpha})$ is a valid nonterminal (see item (c)). In analogy to the previous decomposition constructions, the initial nonterminal covers the entire string, the entire hedge, and the entire alignment $\bar{\alpha}$ (see item (d)). Rules (see item (e)) are constructed for all combinations of (i) operator symbols and (ii) nonterminals that comply with: (iii) the LCFRS algebra, (iv) the sDCP algebra, and (v) the alignment algebra.

Construction 6.3.1. Let $\bar{w} \in \Gamma^*$ with $|\bar{w}| = \ell$, let $\bar{\zeta} \in \mathrm{U}_{\Delta}^*$, and let $\bar{\alpha} \colon [\ell] \to \mathrm{pos}(\bar{\zeta})$ be injective, i.e., $(\bar{w}, \bar{\zeta}, \bar{\alpha})$ is a hybrid tree. We construct the RTG $G = (N, \Sigma, S, R)$, called $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$-*decomposition of* $(\bar{w}, \bar{\zeta}, \bar{\alpha})$, using the intermediate sorted sets N^1 (nonterminals of the LCFRS decomposition) and N^2 (nonterminals of the sDCP decomposition).

(a) Let $k \in [k_{\max}]$ and $N_k^1 = \{\langle (i_1, j_1), \ldots, (i_k, j_k) \rangle \mid i_1, \ldots, i_k, j_1, \ldots, j_k \in [\ell]_0, \forall \kappa, \kappa' \in [k] \colon \kappa \neq \kappa' \text{ implies } [i_\kappa + 1, j_\kappa] \cap [i_{\kappa'} + 1, j_{\kappa'}] = \emptyset\}$. We denote

$$\bar{w}[\langle (i_1, j_1), \ldots, (i_k, j_k) \rangle] = (\bar{w}[i_1 : i_1], \ldots, \bar{w}[i_k : j_k]) \ .$$

(b) Let $l \in \mathbb{N}$ and $s_1, \ldots, s_l \in \mathbb{N}$ be such that $(\mathrm{T}_\Sigma)_{s_1 \cdots s_l} \neq \emptyset$. Let $w_1 \in \mathrm{SB}_{s_1}^{\ltimes}(\bar{\zeta})$, $\ldots$, $w_l \in \mathrm{SB}_{s_l}^{\ltimes}(\bar{\zeta})$ be such that, for each $i, j \in [l]$ with $i \neq j$, we have that w_i and w_j are non-overlapping. Then $\langle w_1, \ldots, w_l \rangle \in N_{s_1 \cdots s_l}^2$. We define the notions $\bar{\zeta}[w_o]$ (for each $o \in [l]$) and $\bar{\zeta}[\langle w_1, \ldots, w_l \rangle]$ as in Construction 6.2.2.

(c) Let $\langle (i_1, j_1), \ldots, (i_k, j_k) \rangle \in N_k^1$, $\langle w_1, \ldots, w_l \rangle \in N_{s_1 \cdots s_l}^2$, and let

$$\tilde{\alpha} \colon \bigcup_{\kappa \in [k]} \{r \mid i_\kappa < r \leq j_\kappa\} \to \bigcup_{m \in [l]} \mathrm{hspan}^{\bar{\zeta}}(w_m)$$

be such that, for each $r \in \mathrm{dom}(\tilde{\alpha})$, we have $\tilde{\alpha}(r) = \bar{\alpha}(r)$ and, for each r in $\mathrm{dom}(\bar{\alpha})$ with $\bar{\alpha}(r) \in \mathrm{codom}(\tilde{\alpha})$, we have $r \in \mathrm{dom}(\tilde{\alpha})$. Then

$$\langle (i_1, j_1), \ldots, (i_k, j_k); w_1, \ldots, w_l; \tilde{\alpha} \rangle \in N_{(k, s_1 \cdots s_l)} \ .$$

(d) $S = \langle (0, \ell); (\varepsilon, (\varepsilon, \rightarrow^{\mathrm{len}(\bar{\zeta})})); \bar{\alpha} \rangle$.

(e) R contains each rule of the form

$$B_0 \to \sigma(B_1, \ldots, B_n)$$

where

(i) $\sigma \in \Sigma_{((k_1, s_1^1 \cdots s_{l_1}^1) \cdots (k_n, s_1^n \cdots s_{l_n}^n), (k_0, s_1^0 \cdots s_n^0))}$
for $k_0, \ldots, k_n \in \mathbb{N}$, $l_0, \ldots, l_n \in \mathbb{N}$, $s_1^0, \ldots, s_{l_0}^0, \ldots, s_1^n, \ldots s_{l_n}^n \in \mathbb{N}$, and $n \in \mathbb{N}$ with $\sigma^{\mathcal{A}_3} = \langle u_1, \ldots, u_{k_0}; \xi_1, \ldots, \xi_{l_0}; \alpha \rangle$,

(ii) for each $q \in [n]_0$, $B_q \in N_{(k_q, \bar{s}_q)}$, where B_q is of the form $\langle B_q^1; B_q^2; \tilde{\alpha}_q \rangle$ and

$$\begin{aligned} B_q^1 &= (i_1^q, j_1^q), \ldots, (i_{k_q}^q, j_{k_q}^q) \\ B_q^2 &= ((w_1^{q,1}, \to^{i_1^{q,1}}) \cdots (w_{s_1^q}^{q,1}, \to^{i_{s_1^q}^{q,1}}), (w_0^{q,1}, \to^{i_0^{q,1}})), \\ &\qquad \ldots, ((w_1^{q,l_q}, \to^{i_1^{q,l_q}}) \cdots (w_{s_{l_q}^q}^{q,l_q}, \to^{i_{s_{l_q}^q}^{q,l_q}}), (w_0^{q,l_q}, \to^{i_0^{q,l_q}})) \ , \end{aligned}$$

(iii) $\bar{w}[\langle B_0^1 \rangle] = \sigma^{\mathcal{A}}(\bar{w}[\langle B_1^1 \rangle], \ldots, \bar{w}[\langle B_n^1 \rangle])$,

(iv) $\bar{\zeta}[\langle B_0^2 \rangle] = \sigma^{\mathcal{A}}(\bar{w}[\langle B_1^2 \rangle], \ldots, \bar{w}[\langle B_n^2 \rangle])$,

(v) and, concerning the alignments, we require that

$$\tilde{\alpha}_0 = \tilde{\alpha}_\sigma \cup \bigcup_{q \in [n]} \tilde{\alpha}_q$$

with

$$\tilde{\alpha}_\sigma = \bigcup_{m \in [l_0]} \{(i_q^0 + \mathrm{off}_{\mathrm{s}}^q(j)) \mapsto (w_0^{0,m} \cdot \mathbf{t}\mathrm{off}_{\theta_m, \mu, \varrho}^{\xi_m}(p) \cdot \diamond) \mid ((q, j) \mapsto (m, p)) \in \alpha\}$$

where for each $q \in [n]$:

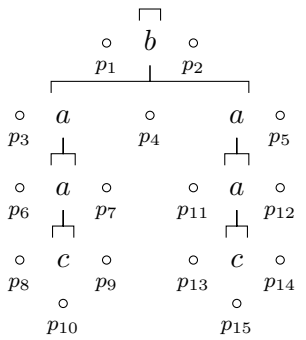
$$\begin{aligned} \lambda_\kappa^q &= j_\kappa^q - i_\kappa^q && \text{for each } \kappa \in [k_q] \\ \mu_r^q(\theta) &= i_0^{q,r} && \text{for each } r \in [l_q] \text{ and } \theta \colon Y_{s_r^q} \to \mathbb{N} \\ (\varrho_r^q(\theta))(o) &= w_o^{q,r} && \text{for each } r \in [l_q],\ \theta \colon Y_{s_r^q} \to \mathbb{N}, \text{ and } o \in [s_r^q] \end{aligned}$$

and for each $m \in [l_0]$:

$$\theta_m = \{y_o \mapsto i_o^{0,m} \mid o \in [s_m^0]\} \ . \qquad \square$$

We observe that because of the restrictions in (c), for each pair of nonterminals $\langle B_1 \rangle$ and $\langle B_2 \rangle$ with $\langle B_1 \rangle \in N_1$ and $\langle B_2 \rangle \in N_2$ at most one nonterminal of the form $\langle B_1, B_2, \tilde{\alpha} \rangle$ can be constructed. This allows an implementation to represent $\tilde{\alpha}$ only implicitly. Next we illustrate the construction with an example.

$\bar{w} = {}_0b_1c_2a_3a_4a_5a_6c_7$



$p_1 = \varepsilon$
$p_2 = \rightarrow$
$p_3 = \downarrow$
$p_4 = \downarrow\rightarrow$
$p_5 = \downarrow\rightarrow^2$
$p_6 = \downarrow^2$
$p_7 = \downarrow^2\rightarrow$
$p_8 = \downarrow^3$
$p_9 = \downarrow^3\rightarrow$
$p_{10} = \downarrow^4$
$p_{11} = \downarrow\rightarrow\downarrow$
$p_{12} = \downarrow\rightarrow\downarrow\rightarrow$
$p_{13} = \downarrow\rightarrow\downarrow^2$
$p_{14} = \downarrow\rightarrow\downarrow^2\rightarrow$
$p_{15} = \downarrow\rightarrow\downarrow^3$

$\bar{\alpha}$
$1 \mapsto p_{13}\diamond$
$2 \mapsto p_8\diamond$
$3 \mapsto p_{11}\diamond$
$4 \mapsto p_4\diamond$
$5 \mapsto p_3\diamond$
$6 \mapsto p_6\diamond$
$7 \mapsto p_8\diamond$

Figure 6.2: A hybrid tree $(\bar{w}, \bar{\zeta}, \bar{\alpha})$ with indication of string positions and $\mathrm{spos}(\bar{\zeta})$.

Example 6.3.2 (Example 4.4.2 cont'd)**.** We construct the reduct for the LCFRS/sDCP hybrid grammar from Example 4.4.2 and the hybrid tree $(\bar{w}, \bar{\zeta}, \bar{\alpha})$ shown in Figure 6.2. To this end, we combine the decomposition G_1 for the LCFRS algebra and the decomposition G_2 for the sDCP algebra given in Example 6.1.4 and Example 6.2.3, respectively.

The initial nonterminal is $\langle (0,7)\ ;\ (\varepsilon, \rightarrow)\ ;\ \bar{\alpha}\rangle$. For the rules we use pairs of rules with the same terminal symbol constructed in Example 6.1.4 and Example 6.1.4 as candidates. We pairwise combine the nonterminals in these rules while adding the unique alignment function $\tilde{\alpha}$ – subject to condition (c). Finally we also check whether the operations of the alignment algebra are compatible with parts the hybrid tree in consideration.

- For rules containing σ, there is at most one extension for each rule in G_1 because $\bar{\alpha}$ is bijective. Let $i, j, k, l \in [7]_0$ such that $0 \leq i \leq j \leq k \leq l$. If there is $p \in \mathrm{spos}(\zeta)$, $i_1, j_1, j_2 \in \mathbb{N}$, and $p' \in \{\downarrow, \rightarrow\}^*$ such that
  - $\mathrm{boundary}^{\zeta}(\{\bar{\alpha}(r) \mid i < r \leq j\}) = (\varepsilon, (p, \rightarrow^{i_1}))$,
  - $\mathrm{boundary}^{\zeta}(\{\bar{\alpha}(r) \mid j < r \leq l\}) = (\varepsilon, (pp', \rightarrow^{j_1+j_2}))$, and
  - $((p', \rightarrow^{j_1,j_2}), (p, \rightarrow^{i_1})) \in \mathrm{SB}^{\ltimes}_{(1)}(\zeta)$,

  then we construct the rule

$$\begin{aligned}\langle (i,l)\ ;\ (\varepsilon, (p, \rightarrow^{i_1}))\ ;\ \tilde{\alpha}\rangle \\ \rightarrow \sigma(\ \langle (j,k), (k,l)\ ;\ (\varepsilon, (pp', \rightarrow^{j_1})), (\varepsilon, (pp' \rightarrow^{j_1}, \rightarrow^{j_2}))\ ;\ \tilde{\alpha}_1\rangle\ , \\ \langle (i,j)\ ;\ (\varepsilon, (p, \rightarrow^{i_1}))\ ;\ \tilde{\alpha}_2\rangle\ )\end{aligned}$$

- for γ we construct all rules of the form

$$\langle (i_1, j_1), (i_2, j_2) \ ; \ (\varepsilon, (p_i, \rightarrow)), (\varepsilon, (p_j, \rightarrow)) \ ; \ \tilde{\alpha} \rangle$$
$$\rightarrow \gamma(\ \langle (i_1, j_1 - 1), (i_2 + 1, j_2) \ ; \ (\varepsilon, (p_i \downarrow, \rightarrow)), (\varepsilon, (p_j \downarrow, \rightarrow)) \ ; \ \tilde{\alpha}_1 \rangle \ )$$

  where, in order to ensure that string and hedge ranges are non-overlapping, we restrict (i_1, j_1), (i_2, j_2), p_i, and p_j as follows.

i_1		j_1	$(p_j, \rightarrow)$		i_2		j_2	$(p_i, \rightarrow)$
1	–	3	$(p_{11}, \rightarrow)$		2	–	7	$(p_{11}, \rightarrow)$
1	–	4	$(p_4, \rightarrow)$		3	–	7	$(p_4, \rightarrow)$
1	–	5	$(p_3, \rightarrow)$		4	–	7	$(p_3, \rightarrow)$
1	–	6	$(p_6, \rightarrow)$		5	–	7	$(p_6, \rightarrow)$

  The lines mark combinations that are admissible according to the sDCP algebra (i.e., hedges do not overlap). The positions $p_8\diamond$ and $p_{13}\diamond$ lay always within the generated hedges, thus, by alignment constraints, also string positions 2 and 7 need to be covered. Since j_1 marks the right end of a string span, the interval (i_1, j_1) cannot cover 7. Likewise, i_1 marks the left end of an interval and, thus, (i_2, j_2) cannot cover position 2. Thus $i_1 = 1$ and $j_2 = 7$. Since the spanned string positions may not intersect another, the dashed combinations are ruled out. Hence, only four rules remain.

- for δ there are 2 rules in each of G_1 and G_2. From the four possible combinations only two are compatible with $\bar{\alpha}$:

$$\langle (1, 2), (6, 7) \ ; \ (\varepsilon, (p_{13}, \rightarrow)), (\varepsilon, (p_8, \rightarrow)) \ ; \ \{2 \mapsto p_{13}\diamond, 7 \mapsto p_8\diamond\} \rangle \rightarrow \delta(\ )$$
$$\langle (6, 7), (1, 2) \ ; \ (\varepsilon, (p_8, \rightarrow)), (\varepsilon, (p_{13}, \rightarrow)) \ ; \ \{2 \mapsto p_{13}\diamond, 7 \mapsto p_8\diamond\} \rangle \rightarrow \delta(\ )$$

- for β there is only one rule in the LCFRS and sDCP composition, respectively. Using the alignment function $\tilde{\alpha}_\beta = \{(0 + 1) \mapsto p_1 \cdot \varepsilon \cdot \diamond\}$, we obtain

$$\langle (0, 1) \ ; \ ((p_3, \rightarrow^2)) \ ; \ \{1 \mapsto p_1\diamond\} \rangle \rightarrow \beta(\ ) \ . \qquad \square$$

Conjecture 6.3.3. Let $(\bar{w}, \bar{\zeta}, \bar{\alpha})$ and G be as in Construction 6.3.1. Let $k, l \in \mathbb{N}$, $s_1, \ldots, s_l \in \mathbb{N}$, $(B^1; B^2; \tilde{\alpha}) \in N_{(k, s_1 \cdots s_l)}$ with

$$B^1 = (i_1, j_1), \ldots, (i_k, j_k)$$
$$B^2 = w^1, \ldots, w^l$$

where, for each $m \in [l]$:

$$w^m = ((w_1^m, \rightarrow^{i_1^m}) \cdots (w_{s_m}^m, \rightarrow^{i_{s_m}^m}), (w_0^m, \rightarrow^{i_0^m})) \ ,$$

and $t \in (\mathrm{T}_\Sigma)_{(k, s_1 \cdots s_l)}$. The following are equivalent:

(1) (1a) $[\![t]\!]_k^{\mathcal{A}_1} = \bar{w}[\langle B^1 \rangle]$

(1b) $[\![t]\!]_{s_1 \cdots s_l}^{\mathcal{A}_2} = \bar{\zeta}[\langle B^2 \rangle]$

(1c) $[\![t]\!]_{(k,s_1 \cdots s_l)}^{\mathcal{A}_3} = (\lambda, \mu, \varrho, \alpha)$ where:
for each $z \in \mathrm{dom}(\tilde{\alpha})$ there is $m \in [l]$ and $\kappa \in [k]$ such that $i_\kappa < z \leq j_\kappa$ and

$$p_0^m \cdot \alpha_m(\theta_m)(\kappa, z - i_\kappa) = \tilde{\alpha}(z)$$

where $\theta_m(y_o) = i_o^m$ for each $o \in s_m$; and
for each $m \in [l]$ and $(\kappa, j) \in \mathrm{dom}(\alpha_m(\theta_m))$, we have that $i_\kappa + j \in \mathrm{dom}(\tilde{\alpha})$.

(2) $\exists d \in (\mathrm{T}_R)_B$ such that $[\![d]\!]_B^{\Pi} = t$. □

We do not give a formal proof for this statement. Soundness, in particular (2) implies (1a) and (1b) should follow from the proofs of Lemma 6.1.2 and Lemma 6.2.4 with slight variations. Concerning (2) implies (1c), we refer to Theorem 4.3.7 and Theorem 4.3.8, which state that the alignment algebra aligns positions in tuples of strings with positions in tuples of contexts in a way that is compatible to the LCFRS and sDCP algebra.

Corollary 6.3.4. Let $(\bar{w}, \bar{\zeta}, \bar{\alpha})$ and G be as in Construction 6.3.1. If Conjecture 6.3.3 holds, then

$$\{t \in (\mathrm{T}_\Sigma)_{(1,(\varepsilon,0))} \mid [\![t]\!]_1^{\mathcal{A}_1} = (\bar{w}), [\![t]\!]_{(\varepsilon,0)}^{\mathcal{A}_2} = \bar{\zeta}, [\![t]\!]_{(1,(\varepsilon,0))}^{\mathcal{A}_3} = (\lambda, \mu, \varrho, \alpha), \alpha_1(\emptyset)(1) = \bar{\alpha}\}$$
$$= \{[\![d]\!]_S^{\Pi} \mid d \in (\mathrm{T}_R)_S\} \ .$$ □

If the precondition holds, then the corollary establishes that Construction 6.3.1 is a correct decomposition for hybrid grammars.

Complexity. The hybrid grammar decomposition uses nonterminals composed from those from the LCFRS decomposition and the sDCP decomposition but adds an alignment function $\tilde{\alpha}$. Recall that $\tilde{\alpha}$ is completely determined by the string ranges. Hence the total number of nonterminals is bounded by the product of the total numbers of nonterminals in the LCFRS decomposition and the total number of nonterminals in the LCFRS decomposition. The same holds for the total number of rules. Thus:

$$|N| \leq |\bar{w}|^{2 \cdot k_{\max}} \cdot |\mathrm{spos}(\bar{\zeta})|^{2 \cdot l_{\max} \cdot (s_{\max}+1)} \quad \text{and}$$
$$|R| \leq |\bar{w}|^{(n_{\max}+1) \cdot k_{\max}} \cdot |\mathrm{spos}(\bar{\zeta})|^{(n_{\max}+1) \cdot l_{\max} \cdot (s_{\max}+1)} \ .$$

7 Discontinuous parsing with split/merge-refined grammars

The formal specification of grammar formalisms and algorithms for induction and parsing as well as the investigation of their theoretical properties are an intellectually stimulating and challenging pursuit on its own. However, the impact and legitimacy of the theory amplify if it is implemented and applied to solve practical problems. This process, should one want to overcome the state of prototypical implementations only capable of handling toy examples, is often connected with numerous challenges. On the one hand, an implementation of algorithms with a higher-polynomial time complexity that still work with large data sets needs to be fast to allow for experimentation on commodity hardware. Low resource consumption is particularly important due to its reciprocity to the overall number of experiments that are feasible and as a means to reduce the energy-induced CO_2 emissions in the light of the climate crisis (cf. Strubell, Ganesh, and McCallum 2019). On the other hand, we are turning from abstract examples to natural languages and face a bunch of problems all connected to data sparsity, i.e., we see only a very small fraction of the set of all possible sentences. Hence, we dedicate this chapter to the description of the implementation of a parsing framework based on LCFRS, hybrid grammars, and the split/merge algorithm as well as its evaluation on a range of data sets. Parts of the material presented in this chapter have been published at COLING 2018 (cf. Gebhardt 2018) and IWPT 2020 (cf. Gebhardt 2020).

Although we experimented with grammar-based dependency parsing at an earlier stage of the development process of our implementation, this thesis will only consider the case of constituent parsing. We made this choice because of the non-negligible amount of time and resources required to fine-tune hyperparameters for new datasets and preparing a thorough evaluation. Also in the light of a wide range of existing accurate and fast dependency parsing approaches the investigation of our presumably slow grammar-based approach appears less worthwile. For first results on dependency parsing with unrefined hybrid grammars we refer to Gebhardt, Nederhof, and Vogler (2017). Note also that the software is in principle prepared for running experiments with split/merge-refined hybrid grammars for dependency parsing.

The chapter is structured as follows: first, we outline requirements that we want our

parser to meet and recall metrics used to compare different systems. We give a brief overview of the software `panda-parser`[1] that implements the grammar formalisms and algorithms described in earlier chapters. Next, we describe a selection of important practical issues and how we addressed them in order to make an evaluation feasible. Later, we list the datasets, standard splits, and the methodology used to evaluate the significance of our empirical results. In terms of the actual experiments, we first describe properties of the grammars that we induce from the data using the algorithms in Chapter 5. Then we investigate the process of split/merge refinements of these grammars and have a brief look into how different substates specialize. Finally, we compare our results with those reported for other systems for discontinuous parsing.

7.1 Requirements of a syntactic parser

First, we collect a range of requirements that the syntactic parser has to meet.

Accuracy. The parser shall assign to each input sentence the *correct* parse tree. Due to syntactic ambiguity, there might be multiple plausible parse trees if a sentence is considered in isolation without its temporal and topical context. Nevertheless, to enable simple and automated evaluation, it is common practise to compute the accuracy of a parser by choosing a particular test set of sentences T and comparing the parser's output for the sentences in T with the hand-annotated, so-called *gold* parse trees for T.

How can the accuracy[2] of a parser be computed? One could compare the number of sentences in T for which the correct tree was predicted by the parser. This metric is called *exact match*. It has the disadvantage that a single wrong label in an otherwise correct parse tree gives the same score as predicting a tree that has no resemblance to the correct one. To this end, Black et al. (1991) proposed the *PARSEVAL* measure that computes recall, precision, and F1-score on labeled constituents. A *labeled constituent*[3] is a pair consisting of a constituent label (i.e., the label of an inner node) and a set of sentence positions that is covered by the corresponding node.

Definition 7.1.1. Let $h = (s, \xi, \alpha)$ be a hybrid tree. The set of *labeled constituents of* h is $\mathrm{lc}(h) = \{(\xi(p), \mathrm{cover}^h(p)) \mid p \in \mathrm{pos}(\xi), \mathrm{children}^\xi(p) \neq \varepsilon\}$. □

Definition 7.1.2. Let A and B be sets. We define the *precision of* A *given* B as

$$\mathrm{Prec}(A \mid B) = \frac{|A \cap B|}{|A|}$$

[1] `https://github.com/kilian-gebhardt/panda-parser`

[2] In this work the term *accuracy* is used in a colloquial sense referring to the quality of the parses when compared to the gold standard.

[3] We use a generalized definition that is compatible with discontinuous trees.

and the *recall of A given B* as

$$\mathrm{Rec}(A \mid B) = \frac{|A \cap B|}{|B|} \ .$$

The F1-score is the harmonic mean of precision and recall:

$$\mathrm{F1}(A \mid B) = \frac{2 \cdot \mathrm{Prec}(A \mid B) \cdot \mathrm{Rec}(A \mid B)}{\mathrm{Prec}(A \mid B) + \mathrm{Rec}(A \mid B)} \ . \qquad \square$$

Let $T = \{s_1, \ldots, s_n\}$ be the set of test sentences. Let $S = \{h_1, \ldots, h_n\}$ be the set of hybrid trees computed by our parser, where h_i is the parse tree for s_i for each $i \in [n]$. Likewise, let $G = \{h'_1, \ldots, h'_n\}$ be the set of gold parse trees, where h'_i is the gold parse tree for s_i for each $i \in [n]$.

Labeled precision, *labeled recall*, and the *labeled F1-score* are computed with respect to the sets $\bar{S} = \{(i, c) \mid i \in [n], c \in \mathrm{lc}(h_i)\}$ and $\bar{G} = \{(i, c) \mid i \in [n], c \in \mathrm{lc}(h'_i)\}$, i.e., we compute $\mathrm{Prec}(\bar{S} \mid \bar{G})$, $\mathrm{Rec}(\bar{S} \mid \bar{G})$, and $\mathrm{F1}(\bar{S} \mid \bar{G})$, respectively. Intuitively, labeled precision is the percentage of labeled constituents hypothesized by the parser which is correct. Labeled recall is the percentage of correct constituents that has been predicted by the parser.

We use the implementation of these measures as provided in `discodop`[4] (van Cranenburgh, Scha, and Bod 2016) using the `proper.prm`[5] parametrization if not declared otherwise. For some evaluations on the TiGer SPMRL corpus (see below) we use the `spmrl.prm`[6] parametrization. The consequences of the parametrizations include that nodes bearing certain labels, such as punctuation, POS tags, and artificial root labels (e.g., `VROOT` or `ROOT`), are ignored. `discodop` also allows for computing labeled recall, precision, and F1-score of just discontinuous constituents. Reporting these scores separately became common in recent years, because although discontinuous constituent occur in about a quarter of the sentences of different treebanks, they have only a small percentage in the total number of constituents.

Speed/resource consumption. The parser shall compute a parse tree in reasonable time for each sentence. It is debatable which parse times are acceptable and dependent on the use case. In this work we are interested in parsers based on LCFRS which are known to be slow compared to greedy transition-based parsers. Building a fast parser is, hence, only a subordinate objective of this thesis. Usually, total or average parse times per sentence for the test set are reported. Also, the average parse time for a sentences of particular length can be of interest as the parse time is often non-linear in the sentence length.

[4] `https://github.com/andreasvc/disco-dop/`

[5] `https://raw.githubusercontent.com/andreasvc/disco-dop/master/proper.prm`

[6] `https://raw.githubusercontent.com/mcoavoux/mtg/master/mind_the_gap_v1.0/data/spmrl.prm`

Next to the parse time one may also take the memory requirements of parsing into account. Again, chart parsers based on LCFRS have a high-memory footprint due to the large number of discontinuous items which can be deduced. The memory requirements are also dependent on the size of the grammar. Therefore, we report sizes of grammars in terms of numbers of rules and nonterminals as well as the size of the grammar after several split/merge cycles in megabytes[7].

Lastly, also time and memory usage of the parser during training is of interest. In theory, these are a one-time costs but if the parser shall be rerun in a different setting, then it can be necessary to optimize hyperparameters anew. This can require a non-trivial number of training runs.

Robustness. A parser that is applied shall ideally assign each sentence it encounters a reasonable syntactic structure. "Robustness is about exploring all constructions humans actually produce, be they grammatical, conformant to formal models, frequent or not." (Chanod 2001). In this thesis, we only evaluate if each sentence of the development set and the test set can be assigned a parse tree and if this tree is of acceptable quality. In particular, the number of parse failures is measured and the recall on labeled constituents is zero for sentences without a parse tree.

Interpretability. Ideally, a parser is also informative with respect to the structural properties of the language it was trained on. For instance, one could analyse variations in the rule weights depending on the latent splits and if there are patterns that align with syntactic theory developed by linguists. However, such comparisons often need to be done in a manual way and can therefore be based only on a limited subset of the induced grammar (if the grammar is large).

7.2 Implementation overview

In this section we describe `panda-parser`[8] which was mainly developed by the author. The software `panda-parser` substantially extends the code[9] provided by Nederhof and Vogler (2014) and contains contributions by Markus Teichmann, Johann Seltmann, and Kevin Mittlöhner.

It provides the following functionality:

- induction algorithms for LCFRS/sDCP hybrid grammars from corpora in different formats (constituent corpora in the NeGra export format and the TiGer XML format, dependency corpora in CoNLL format)

[7] The grammar, the lexicon, and probability assignments are compressed using `gzip`.
[8] `https://github.com/kilian-gebhardt/panda-parser/`
[9] `http://mjn.host.cs.st-andrews.ac.uk/code/coling2014.zip`

- algorithms to compute the chart of an LCFRS, an sDCP, and an LCFRS/sDCP hybrid grammar for a string, an unranked tree, and a hybrid tree, respectively
- the split/merge algorithm for IRTG including EM-training and support for Dirichlet priors ≥ 1
- adapters to the LCFRS parser discodop[10], the run-time of the Grammatical Framework[11], and the finite state toolkit OpenFST[12] to facilitate efficient parsing of strings
- different parsing objectives for split/merge-refined IRTG; in particular: Viterbi parsing, reranking of parses of some baseline IRTG with a split/merge-refined IRTG, variational parsing, max-rule-product parsing
- scripts to run reproducible end-to-end experiments for refinement of LCFRS and LCFRS/sDCP hybrid grammars

Less computationally expensive parts of the software and its command line interface are written in the programming language `Python3.7`. Computationally intensive functionality, such as the computation of charts for hybrid grammars, split/merge training, and the parsing objectives, is written in `C++`[13] where interfaces to the Python codebase are realized with `Cython`. Specifically, we adopt the representation of Cohen et al. (2012) and store the weights of the split/merge-refined versions of a rule $A_0 \to \sigma(A_1, \dots A_n)$ in a single $(n+1)$-dimensional tensor: if for each $i \in [n]$ the number of splits for A_i is l_i, then in the refined grammar we only store the unrefined rule and a *tensor* $w\colon [l_0] \times \cdots \times [l_n] \to [0,1]_{\mathbb{R}}$. In order to process these tensors we utilize the *Eigen* template library[14]. Currently the maximum supported rank n is 7 – supporting larger ranks is a matter of adding boilerplate code.

7.3 Practical issues

In principle, the theory outlined in the previous chapters can be translated very directly into algorithms for extracting grammars from treebanks, for training the grammar's weights, and for selecting parse trees given some input. However, there are limitations due to computational complexity and data sparsity that we need to resolve in order to obtain a parser that works in practise. In this section we discuss LCFRS parsing complexity and several mechanisms to deal with rare words as instances of those limitations.

[10] van Cranenburgh, Scha, and Bod (2016), `https://github.com/andreasvc/disco-dop`.
[11] Ranta (2011), `https://www.grammaticalframework.org/`.
[12] Mohri, Pereira, and Riley (2000), `http://www.openfst.org/`.
[13] `https://github.com/kilian-gebhardt/panda-parser-backend`
[14] `http://eigen.tuxfamily.org/index.php?title=Main_Page`

7.3.1 Coarse-to-fine parsing for LCFRS with CFG

State-of-the-art algorithms for LCFRS parsing, when used with automatically extracted grammars, rely on a coarse-to-fine pipeline and pruning. We want to mention two implementations that use different variants of this approach. On the one hand, there is `disco-dop` by van Cranenburgh, Scha, and Bod (2016), which is written in Python with C++ and Cython backends. Here, a context-free approximation of a given LCFRS, originally introduced by Barthélemy et al. (2001), is computed by splitting each nonterminal A of fanout $k > 1$ into nonterminals $A^*_1, \ldots, A^*_k$. Moreover, the set of rules is changed, e.g., the rule

$$A \to \langle a\ x^1_1 x^2_2, x^1_2\ b\ x^2_1 \rangle (A, C)$$

is replaced by the rules

$$A^*_1 \to \langle a\ x^1_1\ x^2_1 \rangle (A^*_1, C^*_2) \qquad \text{and} \qquad A^*_2 \to \langle x^1_1\ b\ x^2_1 \rangle (A^*_2, C^*_1)$$

and in consequence a superset approximation of the original LCFRS is obtained. The input sentence w is parsed with this PCFG. The chart items corresponding to the n-best derivations are used as a *whitelist* for a subsequent LCFRS parsing step: Only if $A^*_1, \ldots, A^*_k$ with spans $s_1, \ldots, s_k$, respectively, occur in the whitelist, then the item $(A, \bigcup_{i=1}^{k} s_i)$ will be added to the chart.

On the other hand, there is the automata-based framework `rustomata` by Ruprecht and Denkinger (2019). At its core is the idea to use a Chomsky-Schützenberger representation of the given LCFRS: the language of each LCFRS G can be characterized as the homomorphic image $h(R \cap D)$ of the intersection of a regular language R and a multiple Dyck language D. This motivates a parsing pipeline, where a cascade of functions is applied: first the inverse of h is used to generate candidate bracket words for some input w, then bracket words not in R are filtered out, then bracket words not in a context-free approximation of D are filtered out, before only bracket words in D are returned. These bracket words correspond exactly to the derivation trees of G. In this process bracket words are generated in order of descending weights. Also a thresholds can be included to reduce the number of bracket words that are passed to the next stage.

Our experiments employ `disco-dop` to compute the charts for LCFRS because it was already available when we implemented the software and easy to utilize as it is written in Python too. However, we also support the LCFRS parser by Angelov and Ljunglöf (2014), which uses an incremental, heuristic-backed parsing strategy. Yet we found it to be slower in early experiments. We use the list of the 50,000 best PCFG derivations to compute the whitelist for the LCFRS stage.

In order to use `disco-dop` we need to provide an approximating CFG G'. The construction of G' from an LCFRS G by Barthélemy et al. (2001) is without weights, i.e., it does not specify how the probabilities of the rules of the CFG can be obtained.

Providing a probability assignment is non-trivial: observe that G' generates a superset of the language $\mathcal{L}(G)$ and in general it cannot be avoided that some probability mass is redistributed to $\mathcal{L}(G') \setminus \mathcal{L}(G)$. The probability assignment which we use in the implementation assigns each rule resulting from a split up the same probability as it's origin. This penalizes derivations of G' that contain simulations of discontinuous rule applications: if a nonterminal with fanout k is rewritten according to a rule with probability w, then the context-free approximation will contain k applications of rules resulting from the split-up, each with probability w. As $w \in [0,1]_{\mathbb{R}}$ we have that $w^k \leq w$.

A solution might be the assignment of $\sqrt[k]{w}$ to the rules resulting from the split up, but as $\sqrt[k]{w} \geq w$ we loose properness of G'. Viterbi and n-best parsing with such a grammar is still feasible because $1 \geq \sqrt[k]{w}$, i.e., G' is still monotonous. However, `disco-dop` currently rejects those grammars. Van Cranenburgh, Scha, and Bod (2016) circumvent this problem by applying the context-free transformation directly to the discontinuous treebank. This way they obtain a continuous treebank from which they read off a PCFG where each rule's probability equals its relative frequency in the treebank.

7.3.2 The lexical layer and rare words

A frequent problem in natural language processing is the sparsity of the training data. In particular in morphologically rich languages there is a large number of word forms that occur only seldom in spoken and written language. Nevertheless, we want a statistical parser to handle any plausible sentence even if it contains words the parser was not exposed to during training. These word forms are often referred to as *unknown*, *rare*, or *out-of-vocabulary* (OOV) words. In addition, providing parses for sentences with spelling mistakes may be desirable as well. Solving this issue is a research topic on its own. We list some methods that have been developed:

1. *Unking*. Words that occur with a frequency below a certain threshold are replaced by a fresh `<UNK>` (unknown) token. The underlying assumption of this strategy is that rare words in the training data are similarly distributed as rare words in general.

2. Word signatures[15]. This is a refinement of the unking strategy where multiple clusters of rare words are formed. E.g., one looks if an unknown word is capitalized, consists only of capital letters, contains a digit, consists only of

[15] As used, e.g., in the Stanford parser. See `https://github.com/stanfordnlp/CoreNLP/blob/5fdbfb209069276e95e1765093df9855d2cf2c38/src/edu/stanford/nlp/parser/lexparser/BaseUnknownWordModel.java#L205` and `https://github.com/stanfordnlp/CoreNLP/blob/5fdbfb209069276e95e1765093df9855d2cf2c38/src/edu/stanford/nlp/parser/lexparser/EnglishUnknownWordModel.java#L169`.

digits, ends with a certain suffix, contains a hyphen, as well as a combination of these properties.

3. Brown clusters. Brown et al. (1992) propose a method where a given finite vocabulary is clustered such that the likelihood of a text corpus is maximized where an n-gram model defined over these clusters is considered. In the grammar and in the sentences that are meant to be parsed, terminals are then be replaced by their respective cluster. This method is not able to handle unknown words directly. However, the clusters may be computed from monolingual corpora which are usually several orders of magnitude larger than syntactically annotated corpora.

4. Szántó and Farkas (2014) obtain promising results in the context of projective constituent parsing for morphologically rich languages based on PCFG-LA as follows: First they split the nonterminals corresponding to POS tags in an already trained and refined grammar along morphological features (such as gender, number, and case). The probabilities of lexical rules for these splits are initialized according to the training corpus with gold morphological annotations. The grammar is refined by a few epochs of EM training and using a loss criterion based on likelihood some of these splits are undone again. In consequence, they obtain a good granularity of their POS layer in an automated but linguistically supervised fashion. Additionally, they employ an external lexicon and morphological tagging component.

Dakota (2018) compares different approaches and their parametrization (i.e., unknown word thresholds, suffix lengths, cluster sizes, and preprocessing before clustering) for parsing German. He finds it beneficial to compute brown clusters over texts where each word has been replaced with its lemma, its POS tag, and capitalization information. Moreover, excessive smoothing in the split/merge training of latently annotated PCFG generally improves the robustness but reduces the precision when handling rare words.

In the experiments we mostly stick to a relatively simple unking scheme where words that occur less than 5 times are replaced by an `<UNK>` token. In addition, the training and validation set is augmented with copies of the original sentences where all words are unked. The frequency of those sentences in reduced to 0.1 each. We refer to this process as *lexical back-off*. Moreover, we use so-called *Stanford signatures* as implemented in `disco-dop` for evaluations in one of the parsing scenarios without gold POS information (see next section).

7.3.3 Part-of-speech tags

Each word that occurs in a sentence is assigned a *part-of-speech (POS) tag*. Usually, POS tags form a subset of the set of syntactic categories. In the parse trees they may

occur only directly above a word form and each word form has a POS tag as parent. During parsing, two scenarios are distinguished:

1. The POS tags are part of the input, i.e., each word is already assigned the correct POS tags.

2. The POS tags are not part of the input.

The first case makes parsing slightly easier but is slightly inappropriate in practical settings – usually we don't know the POS tags of some arbitrary input sentence that we like to parse. One can argue that POS tags can be predicted by an external model with reasonable high accuracy (about 95%-98% for many languages, see e.g., Yasunaga, Kasai, and Radev 2018). However, as Manning (2011) points out, POS tags are ambiguous as well. Without considering the syntactic and semantic context of a sentence, we cannot hope to correctly assigning all POS tags and achieve 100% accuracy. On the other hand, having only slightly incorrect POS tags can already lead to *error propagation*, i.e., wrong consequences in the subsequent processing stages are drawn from the incorrect POS tags. Hence, ideally one determines POS tags and the syntactic structure simultaneously.

In our experiments, we either use gold POS tags or assign them by an external tagger called `MATE` (Björkelund et al. 2010). The tagger is trained on the training set (or training and development set) of the respective treebank. Moreover, in scenarios where Stanford signatures are used, we do not take any POS tags as input but jointly predict them with the remaining syntactic structure.

7.4 Datasets, splits, and statistical significance

For our experiments on induction and training of grammars and their application to parsing unseen sentences, we use a range of datasets from different languages. Usually there are standard splits for each dataset into a training, a development, and a test set. The training set shall be used to train the model, e.g., extract rules for a grammar and assign weights to them. The development set is a set for preliminary evaluation with the purpose to choose so-called hyper parameters. In our case, this may involve labeling schemes for nonterminals, terminal labeling strategies and their parameters, or the number of split/merge cycles to apply. Finally, on the test set only the final model (or sometimes a few final models, depending on the research question) are applied to give a true estimate of the performance of the model on unseen sentences.

7.4.1 Statistical significance

Using standard splits of these sets has the advantage that the performance reported for different models is comparable. However, it should also be evaluated if differences

in performance are significant. This holds in particular if the differences or the test set are small. It might be that different systems perform equally well and the measured performance differences are just by chance due to the selection of the test set. To reject this *null hypothesis*, statistical significance tests such as *boot strapping* (Berg-Kirkpatrick, Burkett, and Klein 2012) or *approximate randomization* (Noreen 1989; Yeh 2000) should be used. These tests provide a so-called *p-value*, which is the likelihood that some performance difference is observed given that the null hypothesis is true. Performance differences where the p-value is below 0.05 are considered *significant*. Since we cannot assume that the test section of our corpus is a representative sample we use approximate randomization where we run 100,000 trials. We use a reimplementation of Padó's (2006) `sigf` by Dmitry Ustalov[16].

A related aspect, discussed by Gorman and Bedrick (2019), is that research becomes biased towards a certain standard split if it is used for many years. In particular, certain improvements may not be reproducible if systems are trained and evaluated on different randomized splits. This issue is not addressed in our experimental setup.

Finally, some of the algorithms that we use depend on randomization. In particular, the probability assignment computed by the EM algorithm is only close to a *local* optimum of the likelihood function. Hence, within each split/merge cycle, we use the tie breaking step that adds pseudo-random noise to each rule weight. A random seed controls the sequence of pseudo-random numbers used. In our experiments we mostly report average scores over four random seeds and the uncorrected sample standard deviation. Let n be the number of seeds, $i \in [n]$, and x_i be the score of the experiment with random seed i. Then the *average score* is $\bar{x} = \sum_{i \in [n]} \frac{x_i}{n}$ and the *uncorrected sample standard deviation* is $s_n = \sqrt{\frac{1}{n} \sum_{i \in [n]} (x_i - \bar{x})^2}$, i.e., the square root of the *sample variance*. We use the notation $\bar{x} \pm s_n$ to indicate both values. Note that the number of random seeds, which is restricted by computational constraints, is too small to give reliable estimates of the average score and standard deviation over all possible randomizations. However, early experiments with eight random seeds indicate almost identical average scores (to using just four) and slightly smaller standard deviation for all parsing objectives but Viterbi parsing. Still these numbers should be interpreted with caution.

7.4.2 German corpora

Although the framework that we developed in this thesis is meant to be language independent, the majority of experiments during its development were executed on German data. Due to restrictions in computational resources and time, we did not search the design and parameter space equally intensive for other languages. In particular we use the *NeGra* (Skut et al. 1997) and the *TiGer* (Brants et al. 2004) treebanks that provide syntactic analysis of German newswire. Both treebanks contain

[16] `https://gist.github.com/dustalov/e6c3b9d3b5b83c81ecd92976e0281d6c`

Language	treebank (split)	train	dev	test	POS tags
German	NeGra	1–18,602	19,603–20,602	18,603–19,602	54
German	TiGer (HN08)	$i \mod 10 > 2$	$i \mod 10 = 1$	$i \mod 10 = 0$	54
German	TiGer (SPMRL)	1–40,474	40,475–45,474	45,475–50,474	54
Dutch	Lassy small	52,157	6,520	6,523	12

Table 7.1: Information on the standard splits and the number of POS tags of the corpora used in the experiments.

morphological annotation and provide constituent analyses which are comparably flat (i.e., nodes can have many children) and feature discontinuity. In terms of standard splits, we use the ones provided by Dubey and Keller (2003) for NeGra and Hall and Nivre (2008) (short: HN08) and Seddah, Kübler, and Tsarfaty (2014) (short: SPMRL) for TiGer. Information on both splits is given in Table 7.1. Following the literature, we often report results for the development set for sentence of length up to 40. For TiGer HN08, TiGer SPMRL, and NeGra we abbreviate these development sets by $\text{HN08}^{\text{dev}}_{\leq 40}$, $\text{SPMRL}^{\text{dev}}_{\leq 40}$, and $\text{NeGra}^{\text{dev}}_{\leq 40}$, respectively.

Both, the TiGer and the NeGra corpus use the same set of POS tags: the *Stuttgart Tübingen Tag Set* (STTS, Schiller et al. 1999). Unless explicitly mentioned, we use gold tags in our experiments. For TiGer SPMRL we also use the predicted tags provided with the shared task. For TiGer HN08 we use `MATE` to predict POS tags for the test set after training it on the training and development set. For NeGra we present experiments with Stanford signatures where POS tags are jointly predicted during parsing. Unfortunately, we cannot run these experiments on the other larger corpora at the moment, because way to many fall-back rules are created in this approach. A way to circumvent this problem is to create fall-back rules on demand, which we have not implemented yet[17].

7.4.3 Dutch Lassy corpus

We also evaluate in Dutch using the *Lassy small* corpus (van Noord 2009). We use the version of the corpus and the split by van Cranenburgh and Bod (2013), which is obtained by joining 80-10-10 splits of Lassy small's subcorpora. The sizes of the different sets are shown in Table 7.1. We only run experiments with gold POS tags for this corpus. In comparsion to STTS, the tag set is very small. Hence, it seems likely that we would obtain a higher tagging accuracy with an external tagger than for STTS, but we have not validated this conjecture.

	nonterminals	rules / lexical rules	coverage dev. set (with lex. backoff)	parse fails	F1 (≤40)
LCFRS_{ho}	767	50,153 / 28,080	79.2% (86.9%)	4	68.29
$\text{LCFRS}_{\text{r2}\ell}$	817	49,297 / 28,080	77.4% (84.7%)	4	70.36
$\text{hybrid-left}_{\text{child}}$	83,600	206,418 / 28,080	39.2% (42.6%)	128	62.42
$\text{hybrid-right}_{\text{child}}$	30,281	131,426 / 28,080	48.2% (52.5%)	112	63.37
$\text{hybrid-f1}_{\text{strict-v-1-h-1}}$	6,044	70,739 / 39,094	66.2% (78.1%)	107	71.75
$\text{hybrid-f2}_{\text{child}}$	288	39,123 / 28,080	82.9% (92.5%)	11	63.19
$\text{hybrid-f2}_{\text{strict-v-1-h-1}}$	1,783	63,417 / 39,094	69.3% (81.9%)	108	72.18
$\text{hybrid-f3}_{\text{strict-v-1-h-1}}$	1,394	62,723 / 39,094	69.5% (82.2%)	114	72.22
$\text{hybrid-f2}_{\text{strict}}$	32,281	108,957 / 28,080	46.4% (50.0 %)	166	69.90

Table 7.2: Statistics for the baseline grammars induced from TiGer HN08.

7.5 Properties of baseline grammars

We apply the induction methodology described in Chapter 5 with different parameterizations to obtain various hybrid grammars. We call the resulting grammars, that have not been split/merge-refined yet, *baseline grammars*. Moreover, we extract binary LCFRS with Markovized nonterminal symbols (see Kallmeyer and Maier 2013, for an overview). For *horizontal* and *vertical* Markovization we choose 1 based on early experiments.

Various statistics of the HN08 baseline grammars are listed in Table 7.2. Here hybrid grammars are named "hybrid-rp_λ". "rp" encodes the method with which the recursive partitioning was obtained, e.g., fk, if the recursive partitionings are extracted from the hybrid trees and then transformed to maximum fanout k, and left or right for left-branching and right-branching, respectively. "λ" refers to the nonterminal naming scheme. This can be strict or child labeling, or Markovized strict labeling, in which case the vertical (v) and horizontal (h) Markovization parameters are reported. For each grammar the number of nonterminals, the number of rules, and the number of lexical rules thereof, i.e., rules without right-hand-side nonterminals, are given. Moreover, Table 7.2 states properties of applying this grammar to the development set: Here *coverage* refers to the amount of hybrid trees that is in the language of the respective hybrid grammar. We measure it by computing the chart of the hybrid grammar for each hybrid tree in the development set and checking if its language is non-empty. The coverage percentage with lexical backoff refers to the amount of hybrid trees for which such charts can be computed if all terminals are replaced by an `<UNK>` token. The validation performance that controls early stopping during EM training is measured only on the hybrid trees that are covered with or without lexical backoff. Also, the coverage gives an upper bound on the exact match score

[17] Doing this most likely means sacrificing the modularity of the LCFRS parser.

that can be achieved with any weight assignment.[18] The number of parse failures gives the number of sentences (of length ≤ 40) that are not in the language of the LCFRS-component of the hybrid grammar. Lastly, the labeled F1-scores computed by parsing with the baseline grammars according to the Viterbi objective using gold POS tags are given for sentences up to length 40.[19]

The data in Table 7.2 exhibit different trends: both LCFRS and hybrid grammars in their vanilla form are in principle useful to predict phrase structure trees. Binarized and Markovized treebank LCFRS have less nonterminals and rules than the hybrid grammars with similar Markovization settings (i.e., hybrid-f$k_{\text{strict-v-1-h-1}}$). This lower granularity increases the coverage on the development set and reduces the number of parse fails. However, with finer nonterminals the accuracy of the baseline grammar is improved. Hybrid grammars induced according to the strict and child labeling strategy exhibit higher and lower number of nonterminals, respectively. For strict labeling this leads to comparably high accuracy but reduced coverage. On the other hand, with child labeling the coverage is improved but the accuracy suffers. These phenomena motivate the application of the split/merge algorithm quite well: increasing the number of crisp manual splits will up to some point improve the accuracy of the grammars but at the same time reduce the coverage. However if higher granularity is obtained by splitting states, then they can be interpreted as soft splits and the coverage does not suffer. Still, the increase in nonterminals allows for the rule probabilities to be better fitted to the training data, which to a certain degree should improve the accuracy. Two aspects will be interesting to see: Can the split/merge method make up for the missing crisp splits in the baseline grammars? If so, to which extend does the accuracy of the grammars improve?

7.6 Split/merge refinement

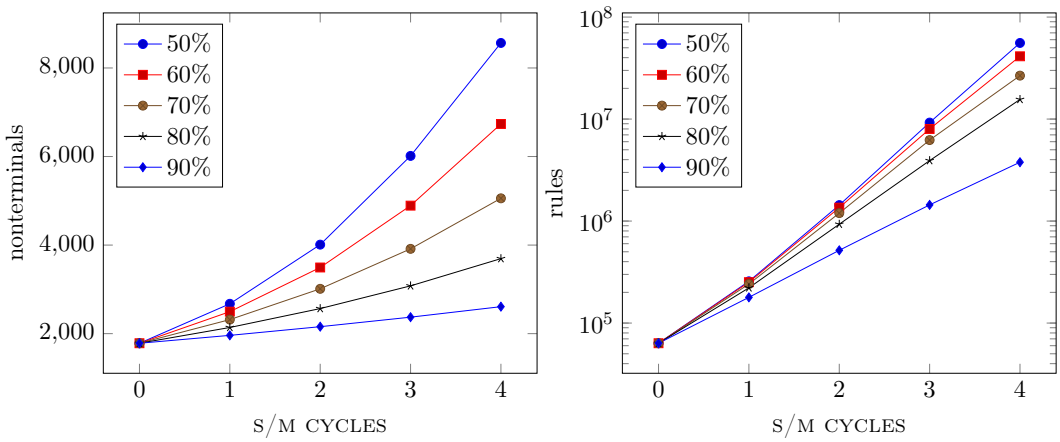
After inducing a grammar, we apply the split/merge procedure to it. The procedure is controlled by the hyperparameters listed in Table 7.3. For some of the hyperparameters we use fixed values: We always stop EM training after 20 epochs at the latest. We stop earlier if the likelihood of the validation set decreases continuously for 6 epochs and reset to the probability assignment where validation likelihood peaked. However, we always apply at least 6 epochs of EM training (3 after smoothing). The smoothing parameter γ is set to 0.01 except for lexical rules where we use 0.1. These values are adopted from Petrov et al. (2006). For the remaining parameters we test different values.

[18] Technically, an exact match as implemented in `discodop` is granted if the sets of labeled constituents of the parser's output and the gold tree are equal. Since certain constituents are ignored according to `proper.prm` this might be the case even if the trees differ slightly.

[19] We just present F1-scores for labeled constituents because precision and recall usually differ at most by a factor of 0.05. Especially for the refined grammars the differences are often smaller.

Parameter	Range
Number of s/m cycles	0 – 8
Merge rate	50% – 90%
EM epochs (fixed)	6 (3 after smoothing) – 20
Smoothing factor γ (fixed)	0.01 (0.1 lexical)
Dirichlet prior	1.01 – 11.0
Random seed	0 – 4

Table 7.3: Hyperparameters for split/merge training and the ranges that we consider.



s/m	50%		60%		70%		80%		90%	
cycl.	nonts	rules	nonts	rules	nonts	rules	nonts	rules	nonts	rules
0	1,783	63,417	1,783	63,417	1,783	63,417	1,783	63,417	1,783	63,417
1	2,674	256,568	2,496	251,606	2,317	241,179	2,139	221,436	1,961	178,728
2	4,010	1,433,971	3,494	1,353,768	3,012	1,196,347	2,566	931,353	2,157	516,813
3	6,014	9,235,766	4,891	7,996,638	3,915	6,223,198	3,079	3,915,165	2,372	1,440,985
4	8,565	55,749,788	6,735	41,243,604	5,056	26,655,311	3,694	15,567,727	2,609	3,776,450
4	352MB		268MB		181MB		113MB		36MB	

Figure 7.4: Size of a hybrid grammar (TiGer HN08, $\text{f2}_{\text{strict-v-1-h-1}}$, prior: 2.0, random seed: 0) depending on the number of split/merge cycles and the merge rate.

objective / merge rate	50%	60%	70%	80%	90%
strict-Markov-v-1-h-1 labeling					
Viterbi	78.63 ±0.18	78.70 ±0.10	79.03 ±0.17	79.17 ±0.20	79.38 ±0.11
variational	80.31 ±0.26	80.15 ±0.18	80.27 ±0.09	80.25 ±0.13	80.21 ±0.13
max-rule-product	**80.54** ±0.15	**80.48** ±0.20	**80.52** ±0.15	**80.54** ±0.08	**80.53** ±0.20
rerank	79.34 ±0.23	79.26 ±0.19	79.26 ±0.09	79.17 ±0.13	79.12 ±0.15
child labeling					
Viterbi	77.96 ±0.24	77.69 ±0.08	77.99 ±0.18	77.78 ±0.07	76.74 ±0.20
variational	78.39 ±0.27	78.16 ±0.23	78.50 ±0.06	78.22 ±0.13	76.92 ±0.27
max-rule-product	78.56 ±0.23	78.28 ±0.15	**78.64** ±0.06	78.37 ±0.14	77.12 ±0.24
rerank	69.93 ±0.05	69.83 ±0.06	69.83 ±0.08	69.60 ±0.07	69.28 ±0.12

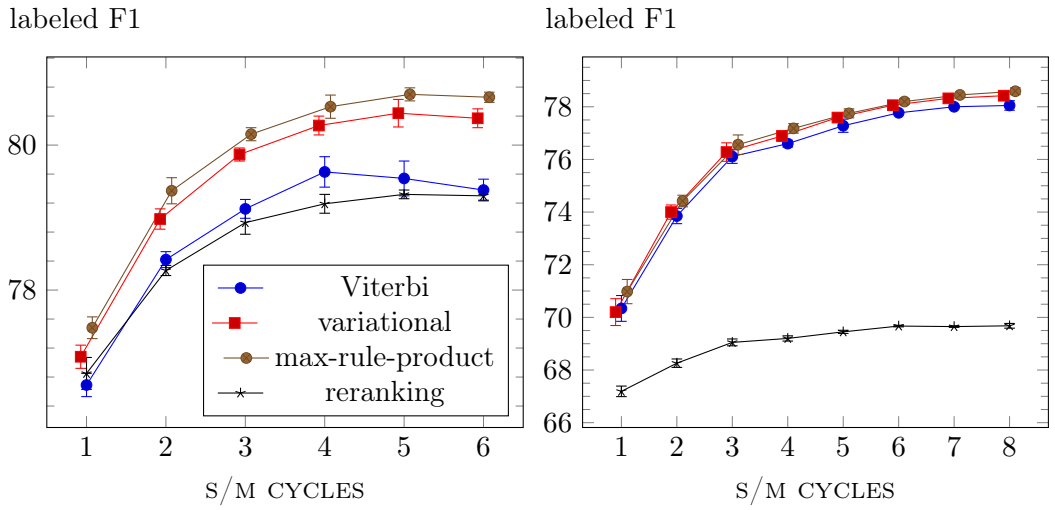
Table 7.5: Average F1-scores on $\text{HN08}^{\text{dev}}_{\leq 40}$ with a hybrid grammar (f2, 4 s/m cycles, prior 2.0, random seeds $\{0, 1, 2, 3\}$) for two nonterminal labeling strategies under different parsing objectives and different merge rates.

7.6.1 Exploratory analysis for TiGer

We use the TiGer corpus in the HN08 split to explore some regions the huge parameter space. Due to computational limitations and the need to run each experiment with multiple random seeds, we cannot explore the entire parameter space in reasonable time. Instead, we carried out a series of experiments where we vary hyperparameters on at a time and observe how this influences the parsing results. Once a hyperparameter has been selected, we maintain it for later experiments. All hyperparameters are listed in the caption of the relevant figures and tables. The initial choice of hyperparameters is based on preliminary experiments. Moreover, we hope to transfer learnings from the TiGer HN08 split to the other corpora and splits with the aim to restrict the search even more in these cases.

Number of split/merge cycles and merge rate. We vary the number of split/merge cycles and the merge rate, i.e., the percentage of splits that is reversed. Clearly, both parameters influence the size of the grammar that is obtained (see Figure 7.4 for an example). For each merge rate, the number of nonterminals and rules grows exponentially in the number of split/merge cycles as expected. However, if 90% of splits are merged again, then already after 3 split/merge cycles the total number of rules is one order of magnitude smaller compared to merging just 50% of the splits. The difference grows even further after another split/merge cycle. Recall that the parsing complexity depends linearly on the size of the grammar in theory. In practise the differences in run time might be even larger due to the cache design of modern computing systems. Thus, smaller grammars obtained by higher merge rates should be preferred if possible without degrading the accuracy of the grammar.

The influence of the merge rate on the accuracy of the resulting grammar for



(a) strict-Markov-v-1-h-1 labeling

(b) child labeling

Figure 7.6: Average F1-scores for different parsing objectives and numbers of s/m cycles at a merge rate of 90%.

Markovized strict labeling is showcased in Table 7.5 (upper part). Interestingly, keeping more of the splits does not improve the quality of the grammars substantially.[20] On the contrary, for the Viterbi objective keeping less splits leads to even higher scores. The latter case might be explained by a reduction in derivational ambiguity such that the best derivation tree is more likely to coincide with the most likely operator and parse tree. In general it might be that the baseline grammar already has a relatively high number of nonterminals and rules and does not require that many splits. To test this hypothesis we run another experiment (see Table 7.5, lower part) with the child labeling strategy, where the baseline grammar has less nonterminals and rules. We see that with merge rates at 80% or 90% the accuracy declines.

Figure 7.6 shows the labeled F1 for different parsing objectives in dependence on the number of split/merge cycles performed. We observe that after initially large improvements the curves flatten. For child labeling the region of convergence is close to the scores obtained with higher merge rates after just 4 split/merge cycles. For Markovized strict labeling we observe a drop in accuracy in the 6th split/merge cycle – for the Viterbi parsing objective this decline starts already after the 5th split/merge cycle. Overfitting after more than 6 split/merge cycles is also observed by Z. Huang and Harper (2009) for PCFG-LA.

[20] The differences for max-rule-product are *not* significant, e.g., $p > 0.74$ (70% vs. 80%) and $p > 0.54$ (60% vs. 80%).

objective / prior	1.0	1.01	1.1	1.5	2.0	3.0	11.0
Viterbi	75.09 ±0.26	77.14 ±0.31	77.70 ±0.38	78.19 ±0.18	78.63 ±0.18	78.86 ±0.14	79.27 ±0.26
variational	79.33 ±0.19	79.69 ±0.03	79.94 ±0.23	80.25 ±0.11	80.31 ±0.26	80.30 ±0.13	79.85 ±0.20
max-rule-product	79.66 ±0.23	79.98 ±0.04	80.22 ±0.27	**80.57** ±0.16	**80.54** ±0.15	**80.59** ±0.17	80.12 ±0.16
rerank	78.60 ±0.14	79.07 ±0.06	79.31 ±0.11	79.40 ±0.09	79.34 ±0.23	79.15 ±0.12	77.96 ±0.09

Table 7.7: Average F1-scores on $\text{HN08}^{\text{dev}}_{\leq 40}$ with a hybrid grammar ($\text{f2}_{\text{strict-v-1-h-1}}$, random seeds $\{0, 1, 2, 3\}$, 4 s/m cycles, merge rate 50%) under different parsing objectives and Dirichlet priors.

Dirichlet priors. Results of an experiment with different Dirichlet priors are shown in Table 7.7. There are two notable trends: the F1-scores improve in all cases but one (reranking with prior 11.0) if a non-trivial prior (i.e., >1) is used. The improvement is highest for the Viterbi objective where each of our increments in the prior parameter leads to a significant improvement of the labelled F1-score. On the contrary, the F1-score degrades if the prior parameter is set to 11.0 for the other objectives. Instead, the prior parameters 1.5, 2.0, and 3.0 exhibit the best scores for the projection-based objectives (differences are not significant). For the reranking objective the values 1.1, 1.5, and 2.0 seem to be optimal. Most of the remaining experiments use the prior value 2.0, which corresponds to increasing the expected frequency of each by 1.0 during EM training.

Parsing objectives. An observation independent of merge rates, priors, nonterminal labeling strategies (see the previous tables) is that the Viterbi and the reranking objective perform worse than the variational and the max-rule-product objective. The exact order of the Viterbi and the reranking objective differs: for grammars derived from a child labeling baseline grammar we note that reranking performs several points in F1 worse than the Viterbi objective. As this order is mostly reversed for (Markovized) strict labeling, we suppose that the reason is that the candidates provided by the child labeling baseline grammar are too weak. To further investigate this hypothesis, we used an oracle reranker (i.e., one that selects the best tree by comparing each tree to the gold tree using the labeled F1-score). In this scenario the labeled F1-scores (on $\text{HN08}^{\text{dev}}_{\leq 40}$ with a fanout 2 hybrid grammar) are 80.69 for child labeling, 90.65 for Markovized strict labeling, and 81.31 for strict labeling. Thus, for both, child and Markovized strict labeling, reranking with a split/merge-refined grammar stays about 10 F1 points below its potential. However, we observe that the child and strict labeling baseline grammars provide candidates that are worse than Markovized strict labeling.

The projection-based objectives are in most cases the superior parsing objectives. Similar to the observation of Petrov and Klein (2007), max-rule-product significantly outperforms the variational objective in terms of labeled F1-score. The explanation of Petrov and Klein (2007) is that the rules favoured in max-rule-product are individually

objective/grammar	$\text{left}_{\text{child}}$	$\text{right}_{\text{child}}$	$\text{f1}_{\text{strict-v-1-h-1}}$	$\text{f2}_{\text{strict-v-1-h-1}}$	$\text{f3}_{\text{strict-v-1-h-1}}$
Viterbi	67.74 ±0.20	71.26 ±0.23	78.36 ±0.17	79.53 ±0.23	79.57 ±0.10
variational	69.23 ±0.14	72.89 ±0.12	79.72 ±0.07	80.35 ±0.08	80.32 ±0.05
max-rule-product	69.40 ±0.16	73.01 ±0.10	80.00 ±0.15	**80.62** ±0.12	**80.54** ±0.03
rerank	66.79 ±0.09	68.85 ±0.04	78.69 ±0.10	79.26 ±0.09	79.23 ±0.04

Table 7.8: Average F1-scores on $\text{HN08}^{\text{dev}}_{\leq 40}$ with different hybrid grammars (4 s/m cycles (left/right), 5 s/m cycles (fk), merge rate 90%, prior 2.0, random seeds $\{0, 1, 2, 3\}$) under different recursive partitioning/labeling strategies.

likely across derivations. This aligns well with the F1-score on labeled constituents. In contrast, the variational objective minimizes KL-divergence to the distribution on complete derivation trees. Hence, one might suspect that the exact match score is higher than with max-rule-product. This is apparently not the case as Table 7.10 shows.

Nonterminal labeling strategy. We return to the question of how to choose the initial nonterminal labeling strategy. The results on the TiGer HN08 development data (cf. Table 7.5) suggests that despite the split/merge refinement, grammars obtained by the child labeling (best average F1: 78.64) stay behind the Markovized strict labeling strategy (best average F1: 80.54). In principle, we can imagine a refinement of the baseline grammar with child labeling that matches exactly the baseline grammar with (Markovized) strict labeling. This grammar could further be refined. Hence, both models are equally potent models at least if the Viterbi decoding strategy is considered. For the other decoding strategies (projections/reranking), the structure of the baseline derivations may play a more important role (cf. the oracle reranking scores above). However, we observe that also with Viterbi decoding, child labeling is surpassed by Markovized strict labeling. One explanation for this might be that there is ambiguity in how to refine the grammar in order to fit the training data well. A finer nonterminal labeling strategy for the baseline grammar resolves some of this ambiguity whereas the split/merge procedure is not able to do this from the limited training data.

Recursive partitioning strategy. Table 7.8 shows results for a selection of hybrid grammars that result from baseline grammars with different recursive partitioning strategies and manually selected nonterminal labeling schemes. Also the number of split/merge cycles was chosen to fit the particular type of grammar well. Notably, the left-branching and the right-branching recursive partitioning strategy does not yield grammars with competitive accuracy even after refinement. For all parsing objectives the accuracy of the fanout-1 grammar is significantly worse than that of the fanout-2 grammar. The differences between fanout-2 and fanout-3 are insignificant.

grammar	LCFRS_{ho}		$\text{LCFRS}_{\text{r2}\ell}$	
prior	1.0	2.0	1.0	2.0
Viterbi	78.48 ±0.29	78.88 ±0.16	78.73 ±0.21	79.40 ±0.26
variational	78.80 ±0.34	79.08 ±0.10	79.09 ±0.20	79.79 ±0.14
max-rule-product	79.10 ±0.37	79.40 ±0.10	79.34 ±0.15	**80.01** ±0.16
rerank	73.85 ±0.32	73.70 ±0.09	74.75 ±0.05	74.96 ±0.09

Table 7.9: Average F1-scores on $\text{HN08}^{\text{dev}}_{\leq 40}$ with different LCFRS (5 s/m cycles).

grammar	f2_{child} (4 s/m cycles, 70%)				$\text{f2}_{\text{strict-v-1-h-1}}$ (5 s/m cycles, 90%)			
objective	EM	disc. F1	Rec	Pre	EM	disc. F1	Rec	Pre
Viterbi	41.98 ±0.17	31.10 ±0.57	20.97	60.13	40.87 ±0.77	45.51 ±1.03	38.09	56.52
variational	43.10 ±0.14	31.43 ±0.64	21.19	60.79	42.73 ±0.32	**47.01** ±0.41	39.22	58.64
max-rule-product	43.16 ±0.19	31.58 ±0.61	21.22	61.74	42.94 ±0.25	**47.28** ±0.44	39.28	59.37
rerank	32.81 ±0.12	26.73 ±0.21	20.10	39.90	42.23 ±0.18	46.48 ±0.98	39.22	57.02

Table 7.10: Average scores for exact match, discontinuous recall, discontinuous precision, and discontinuous F1 of two hybrid grammars on $\text{HN08}^{\text{dev}}_{\leq 40}$ for different parsing objectives.

LCFRS. We also conduct experiments with LCFRS and list results in Table 7.9. While LCFRS outperform hybrid grammars with strict and child labeling, they are inferior to Markovized strict labeling. We observe smaller improvements by using Dirichlet priors in comparison to hybrid grammars. Hence, we mostly focus on hybrid grammars in the following.

Alternative metrics. Finally, we look at the metrics exact match and discontinuous labeled F1. We display the scores for two hybrid grammars and different parsing objectives in Table 7.10. Concerning the different parsing objectives we observe similar rankings as for the labeled F1 metric. For discontinuos F1 the differences between variational and max-rule-product are statistically insignificant while max-rule-product is significantly better than the other objectives. Interestingly, the differences in exact match are very small. Child labeling slightly outperforms Markovized strict labeling in this metric with the exception for the reranking objective, where child labeling scores worse. In terms of discontinuous F1-score, we find that Markovized strict labeling is about 15 points better than child labeling. While the precision on discontinuous constituents is very similar for both grammars, the recall of Markovized strict labeling is almost twice as high as with child labeling. We suspect that statistical learning of splits and rule probabilities is much harder for discontinuous constituents due to their low frequency in the training data. Hence, hard-coded context-based splits could be especially informative in this case.

grammar	F1	*(prod.)*	Disc. F1	*(prod.)*
Likelihood-based training (prior: 1.0)				
LCFRS_{ho}	78.85 ±0.34	79.91	32.57 ±1.91	33.81
$\text{LCFRS}_{\text{r2}\ell}$	79.25 ±0.15	79.93	33.82 ±0.21	34.19
$\text{hybrid-f2}_{\text{child}}$	77.82 ±0.24	78.52	30.64 ±0.28	31.15
$\text{hybrid-f2}_{\text{strict-v-1-h-1}}$	79.66 ±0.23	80.55	45.56 ±0.55	47.65
MAP training (prior: 2.0)				
LCFRS_{ho}	79.03 ±0.42	79.83	30.98 ±0.53	33.00
$\text{LCFRS}_{\text{r2}\ell}$	80.01 ±0.16	81.02	34.15 ±0.22	35.70
$\text{hybrid-f2}_{\text{child}}$	78.56 ±0.23	79.36	32.15 ±0.61	33.36
$\text{hybrid-f2}_{\text{strict-v-1-h-1}}$	80.54 ±0.16	**81.75**	47.32 ±0.96	**49.82**

Table 7.11: (Average) F1-scores on $\text{HN08}^{\text{dev}}_{\leq 40}$ after training for 4 s/m cycles at 50% merge rate using the max-rule-product objective. Columns labeled *(prod.)* show scores for the product models.

Product of latent variable grammars. Finally, we also consider the product parsing objective introduced by Petrov (2010). In Table 7.11 we compare LCFRS and hybrid grammars obtained with Likelihood-based and MAP-based EM training both using individual and product grammars. In each induction/training scenario the score of the product model is higher than the average of the individual grammars – in most cases it is also above the standard deviation. The ranking of the different grammars is maintained when moving to the product model. In particular the $\text{f2}_{\text{strict-v-1-h-1}}$ hybrid grammar shows the highest overall score.

Table 7.11 shows more interesting phenomena: the hybrid grammars are particularly strong in terms of discontinuous F1 when compared to LCFRS. MAP training is not always beneficial as the case of LCFRS_{ho} shows: for three of the considered scores, Likelihood-based training seems to be superior. However, the differences are most promiment when considering discontinuous F1, which tends to have a very high variance. Hence, we surmise that this result is by chance and would not hold up if more samples are used. In summary, we may still conclude that the positive effects of MAP training and product grammars is (mostly) complementary.

7.6.2 Results for TiGer (SPMRL split)

The SPMRL split of the TiGer corpus also uses about 80% of the data for training and 10% for development and testing. Hence, we will just apply the parametrization that we found best for HN08 for this data set. We note that in contrast to HN08 each part of the split is formed of successive sentences. Hence it is quite likely that text in the development and test data covers different topics than the training data.

	≤ 40			all		
objective	F1	F1 (disc)	EM	F1	F1 (disc)	EM
Viterbi	81.62 ±0.14	44.03 ±0.49	46.28 ±0.26	81.23 ±0.12	43.73 ±0.40	45.72 ±0.26
variational	82.35 ±0.20	44.80 ±0.89	47.60 ±0.47	81.96 ±0.17	44.51 ±0.90	47.02 ±0.47
max-rule-product	**82.51** ±0.22	**45.14** ±0.72	**47.74** ±0.51	**82.11** ±0.20	**44.80** ±0.73	**47.16** ±0.51
rerank	81.17 ±0.20	43.74 ±1.00	46.80 ±0.42	80.74 ±0.19	43.41 ±0.93	46.25 ±0.42
			product model			
variational	83.19	46.70	49.10	82.80	46.34	48.50
max-rule-product	83.39	47.21	49.22	82.98	46.84	48.62

Table 7.12: Average scores on $\text{SPMRL}^{\text{dev}}$ with hybrid-$\text{f2}_{\text{strict-Markov-v-1-h-1}}$ (5 s/m cycles, merge rate 90%, prior 2.0, random seeds $\{0, 1, 2, 3\}$) under different parsing objectives.

	nonterminals	rules / lexical rules	coverage dev. set (with lex. backoff)	parse fails	F1
LCFRS_{ho}	716	20,030 / 8,277	88.2% (88.2%)	4	68.88
$\text{LCFRS}_{\text{r2}\ell}$	787	21,243 / 8,277	82.7% (82.7%)	8	70.07
$\text{hybrid-left}_{\text{child}}$	41,363	94,674 / 8,277	44.2% (44.2%)	44	59.84
$\text{hybrid-right}_{\text{child}}$	18,023	63,265 / 8,277	47.9% (47.9%)	38	60.84
$\text{hybrid-f1}_{\text{child}}$	1,086	16,057 / 8,277	90.1% (90.1%)	2	61.37
$\text{hybrid-f2}_{\text{child}}$	279	13,945 / 8,277	93.5% (93.5%)	2	62.06
$\text{hybrid-f2}_{\text{strict-Markov}}$	1,623	33,923 / 19,198	70.0% (81.1%)	42	71.34
$\text{hybrid-f2}_{\text{strict}}$	20,766	50,480 / 8,277	53.9% (53.9%)	32	68.82
$\text{hybrid-f3}_{\text{child}}$	158	13,622 / 8,277	93.5% (93.5%)	2	61.91

Table 7.13: Statistics for the baseline grammars induced from NeGra. Labeled F1-score on $\text{NeGra}^{\text{dev}}_{\leq 40}$.

Table 7.12 shows parsing results with hybrid grammar on the development set. The differences between parsing objectives are similar to what we observed on HN08. The overall scores obtained on this split are higher than those obtained on HN08. Data without length restriction indicates only a small decline in scores, however, only 1.26% of the sentences in the development set are of length ≥ 40.

7.6.3 Results for NeGra

We report results for a reduced set of experiments for the German NeGra corpus. An overview on the properties of the baseline grammars is given in Table 7.13. Notably, the coverage of the left-branching and right-branching grammar and the grammar using the strict nonterminal labeling is low. Early experiments confirmed that these grammars are not interesting for further refinement. Very likely this is a consequence of the smaller size of the NeGra corpus compared to TiGer. We changed the selection

(SM cyc./merge-%)	child labeling					strict-Markov
objective	6/60%	6/70%	6/80%	7/80%	8/90%	5/90%
Viterbi	78.94 ±0.40	79.67 ±0.34	79.89 ±0.61	79.64 ±0.59	80.00 ±0.58	76.11 ±0.36
variational	81.00 ±0.34	81.39 ±0.26	81.49 ±0.12	81.52 ±0.26	81.06 ±0.16	77.38 ±0.25
max-rule-product	81.38 ±0.40	**81.78** ±0.17	**81.76** ±0.07	**81.82** ±0.12	81.28 ±0.32	77.67 ±0.20
rerank	77.43 ±0.23	77.53 ±0.16	77.54 ±0.14	77.42 ±0.28	77.03 ±0.30	77.19 ±0.35

Table 7.14: Average F1-scores on $\text{NeGra}^{\text{dev}}_{\leq 40}$ (hybrid-f2, prior 2.0, random seeds $\{0, 1, 2, 3\}$, gold POS tags) using different nonterminal labeling schemes, numbers of s/m cycles, and merge rates.

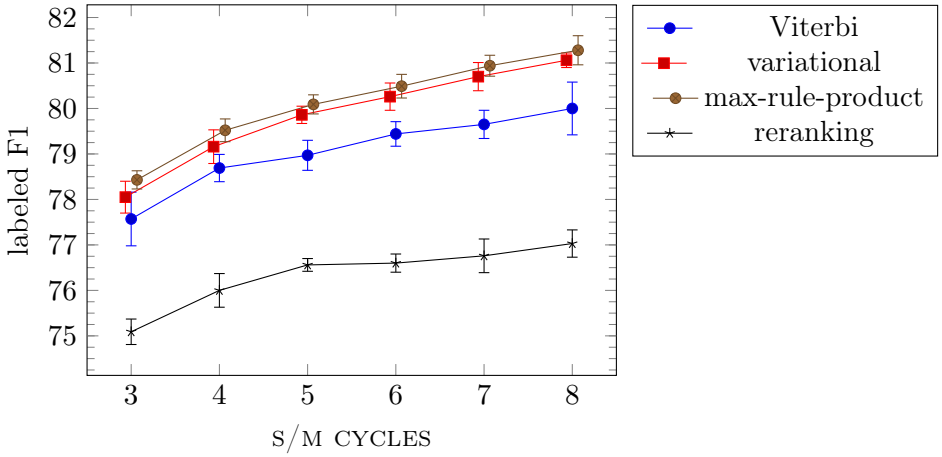


Figure 7.15: Average F1-scores on $\text{NeGra}^{\text{dev}}_{\leq 40}$ (f2_{child}, prior 2.0, merge rate 90%, random seeds $\{0, 1, 2, 3\}$, gold POS tags) for different s/m cycles.

objective / grammar	LCFRS_{ho}	$\text{LCFRS}_{\text{r2}\ell}$
Viterbi	78.21 ±0.45	78.08 ±0.19
variational	79.63 ±0.40	79.63 ±0.14
max-rule-product	80.06 ±0.42	79.84 ±0.20
rerank	78.72 ±0.50	78.99 ±0.14

Table 7.16: Average F1-scores on $\text{NeGra}^{\text{dev}}_{\leq 40}$ with LCFRS (5 s/m cycles, merge rate 90%) and gold POS tags.

objective / merge rate	60%	70%	80%	90 %
Viterbi	77.35 ±0.26	77.52 ±0.15	77.69 ±0.46	76.00 ±0.46
variational	78.11 ±0.39	78.02 ±0.22	78.01 ±0.40	76.74 ±0.40
max-rule-product	**78.86** ±0.38	**78.74** ±0.53	**78.79** ±0.37	77.35 ±0.37
rerank	70.29 ±0.14	70.61 ±0.05	71.19 ±0.16	71.11 ±0.16
POS acc. (mrp)	96.13 ±0.07	96.14 ±0.06	96.20 ±0.05	96.06 ±0.05
	product model (cf. Petrov 2010)			
variational	78.81		78.93	
max-rule-product	**79.41**		79.01	

Table 7.17: Average F1-scores on $\text{NeGra}^{\text{dev}}_{\leq 40}$ (f2_{child}, prior 2.0, random seeds $\{0, 1, 2, 3\}$, 6 s/m cycles, stanford signatures) for different merge rates. The POS tagging accuracy for max-rule-product is given in the fifth row.

of base grammars to contain more grammars child labeling for this reason. However, the differences in coverage and accuracy for grammars with child labeling with fanout 1, 2, and 3 is comparably small, thus, we focus mostly on grammars with child labeling and fanout 2.

In Table 7.14 we vary the merge rate and the number of split/merge cycles only and find 7 cycles with 80% mere rates to yield the best results across parsing objectives for the fanout 2 grammar with child labeling. In contrast to the larger TiGer corpus the accuracy of Markovized strict labeling falls short of child labeling. We suspect that the reason for this is the higher difference in coverage (93.5% vs. 70% for NeGra, 82.9% vs. 69.3% for HN08). Based on evidence from TiGer, we analyse how the grammar improves based on the number of split/merge cycles at a (suboptimal) merge rate of 90%. It seems as if the F1-score slowly approaches the F1-score of the better 80% choice (see Figure 7.15). Similar to the TiGer corpus we obtain lower scores with LCFRS (see Table 7.16) than with hybrid grammars. For LCFRS the differences between the binarization strategies are insignificant. This is in accordance with the observation of Kallmeyer and Maier (2013, Table 2) that Markovized LCFRS obtained with different binarization strategies have a similar performance.

Experiments where we use stanford signatures in the terminal layer and predict POS tags jointly during parsing are shown in Table 7.17. We note again that merge rates of 80% or lower lead to favourable grammars and that max-rule-product is the best of the parsing objectives considered.[21] The POS accuracy for the max-rule-product objective is slightly above 96%, which is worse but in the vicinity of good stand-alone taggers. Overall, the F1-score drops significantly in comparison to scenarios where gold POS tags are available. The differences between the best

[21] Max-rule-product outperforms variational significantly ($p \leq 10^{-5}$); differences for max-rule-product with 60%, 70%, and 80% are not significant ($p > 0.55$).

grammar	F1	*(prod.)*	Disc. F1	*(prod.)*
Likelihood-based training				
LCFRS_{ho}	80.14 ±0.54	81.35	39.57 ±2.00	41.09
$\text{LCFRS}_{\text{r2}\ell}$	78.74 ±0.43	80.25	38.01 ±1.64	39.49
$\text{hybrid-f2}_{\text{child}}$	80.38 ±0.33	82.06	40.73 ±0.86	**44.52**
$\text{hybrid-f2}_{\text{strict-v-1-h-1}}$	77.51 ±0.32	78.17	39.27 ±1.96	42.18
MAP training (prior: 2.0)				
LCFRS_{ho}	80.88 ±0.23	82.27	40.40 ±2.27	41.96
$\text{LCFRS}_{\text{r2}\ell}$	79.38 ±0.79	80.79	40.92 ±2.48	43.04
$\text{hybrid-f2}_{\text{child}}$	81.75 ±0.26	**83.33**	40.31 ±0.68	43.06
$\text{hybrid-f2}_{\text{strict-v-1-h-1}}$	77.62 ±0.12	78.60	40.23 ±0.89	42.52

Table 7.18: (Average) F1-scores on $\text{NeGra}^{\text{dev}}_{\leq 40}$ after training 6 s/m cycles at 80% merge rate. Columns labeled *(prod.)* show scores for the product models.

results in either scenario is about 3 points in labeled F1. It is particularly large for the reranking objective, which indicates that without gold POS tags the baseline grammar is particularly bad at selecting good candidates for the rescoring process. This is supported by an oracle reranking experiment, which yields labeled F1-scores of 90.55 and 86.14 for gold POS tags and jointly predicted POS tags, respectively.

Again, we also consider Petrov's (2010) product of projections objective. Table 7.18 shows results with gold POS tags. In all cases we observe that the product model yields an improvement over the average of the individual grammars. The effect is especially pronounced for the discontinuous F1 scores. However, concerning the discontinuous F1 we also notice that MAP training can fall behind Likelihood-based training. The hybrid grammar with child labeling is the strongest grammar also in the product scenario. In case of joint POS-tag prediction, displayed in the last lines of Table 7.17, the effect of the product model is smaller but in particular for the grammar obtained with a merge rate of 60% there are considerable improvements.

7.6.4 Results for Lassy

To illustrate that our approach split/merge for refined hybrid grammars also generalizes from German to other languages, we do proof-of-concept experiments with the Dutch Lassy treebank. We find that a grammar with the strict-Markov nonterminal labeling work quite well (cf. Table 7.19) whereas the grammar with child labeling has a much lower accuracy. An LCFRS has about 1 point higher scores than the hybrid grammar with child labeling.

Unfortunately, the back-off scheme that is used with stanford signatures leads to grammars that are too large to be handled. Still, these early results are promising and

objective / grammar	$\text{hybrid}_{\text{child}}$	$\text{hybrid}_{\text{strict-Markov-v-1-h-1}}$	LCFRS_{ho}
Viterbi	72.75 ±0.34	79.54 ±0.08	73.67 ±0.10
variational	73.10 ±0.24	80.30 ±0.09	74.10 ±0.05
max-rule-product	73.28 ±0.24	80.67 ±0.11	74.25 ±0.11
rerank	63.07 ±0.11	78.62 ±0.16	66.23 ±0.12
DOP (van Cranenburgh and Bod 2013)		79.0	
DOP (van Cranenburgh, Scha, and Bod 2016)		78.3	

Table 7.19: Average F1-score on the $\text{Lassy}^{\text{dev}}_{\leq 40}$ for hybrid grammars (fanout 2, 90% merge rate, 6 s/m cycles (child), 5 s/m cycles (strict Markov)), LCFRS (90% merge rate, 5 s/m cycles), and results from the literature.

in the vicinity of those by van Cranenburgh and Bod (2013) and van Cranenburgh, Scha, and Bod (2016), who, however, do not use gold POS tags.

7.6.5 Remarks

The experiments on the development set indicate that the split/merge refinement procedure generalizes well for the application of discontinuous parsing with LCFRS and hybrid grammars. There are clear improvements over the unrefined baseline grammars of about 10 points in the F1-score.

We note that the choice of hyperparameters for the split/merge algorithm is important. The values (6 split/merge cycles with 50% merge rate) used by Petrov et al. (2006) are not optimal for our grammars. In particular, we find that the split/merge procedure does not dispense with a careful choice of the nonterminal labeling strategy for the baseline grammar. However, different choices of this strategy may require different hyperparameter. In particular, the merge rate and the number of split/merge cycles should be adjusted as well: too low or too high merge rates can have bad convergence properties and too many split/merge cycles may lead to a decrease of accuracy. Also, we find using a Dirichlet prior beneficial to avoid overfitting.

7.7 Qualitative analysis

We briefly turn to the issue of interpreting the learned grammars. We will only consider POS tags because already the baseline hybrid grammars exhibit too many rules to make a thorough analysis of hybrid grammars feasible. The most interesting grammars for such an analysis are those trained on the NeGra corpus with the Stanford signature scheme. Table 7.20 gives the most probable words for selected POS tags in different refinements.

$(– sentence internal punctuation					KON – coordinating conjunction					
0	)	0.99			0	weder	0.24	sowohl	0.2	
1	-	0.99			1	oder	0.85			
2	’	0.55	"	0.41	2	&	0.24	wie	0.20	als 0.19
3	"	0.79	...	0.17	3	sowie	0.68			
4	/	0.96	...	0.01	4	Und	0.32	Doch	0.24	Aber 0.18
5	"	1.00			5	denn	0.61			
6	(	0.99			6	und	0.96			
					7	sondern	0.77			
					8	aber	0.65	doch	0.19	

NN – common noun								
1	Juni	0.02	Juli	0.002	August	0.001		
2	Telefon.	0.001	Tel.	0.001	Million	0.001	Milliarde	0.001
6	Mark	0.02	Uhr	0.02	Jahren	0.01	Prozent	0.01
8	Millionen	0.009	Milliarden	0.004				

NE – named entity								
5	Frankfurt	0.02	X-en	0.02	Deutschland	0.1	X-rg	0.01
6	dpa	0.009	FR	0.007	AP	0.003	AFP	0.003
9	XXX	0.022	SPD	0.012	CDU	0.012	FR	0.008
11	Bad	0.04	Xxxx	0.009	Peter	0.006	Michael	0.006

PIDAT – attributing indefinite pronoun with determiner								
0	allen	0.24	vielen	0.08	jedem	0.06	wenigen	0.06
1	beiden	0.36	meisten	0.14	solche	0.07	solchen	0.07
2	paar	0.30	wenig	0.10	alles	0.05	bißchen	0.03
3	Alle	0.23	Viele	0.13	Jeder	0.06	Jeden	0.06
4	alle	0.41	viele	0.09	jeden	0.07	jeder	0.06
5	viele	0.28	viel	0.24	solch	0.05	manch	0.03
6	aller	0.28	vieler	0.14	beider	0.05	allen	0.03

ART – determiner								
0	Die	0.47	Der	0.23	Das	0.13		
1	die	0.45	das	0.11	ein	0.10	eine	0.10
2	der	0.65	des	0.27	eines	0.03	einer	0.03
3	der	0.28	den	0.21	die	0.17	dem	0.14

Table 7.20: Most probable words and their probabilities for the splits of selected POS tags on NeGra (hybrid-f2$_{\text{child}}$, 80% merge rate, 6 cycles, seed 0).

For some POS tags (KON, PDS, ART, PIDAT) latent dimensions specialize on capitalization that is required at the beginning of the German sentence. Another trend is that the likelihood-optimizing merger chooses to reserve an exclusive latent dimension for frequent lexical entities (see, e.g., KON, "$("). For "$(" this is also the case with a higher merge rate of 90% while many other lexical categories are less splits in this case.

Some of the latent annotations of NN specialize on nouns with similar properties, e.g., measure words, months, and abbreviations. A similar trend can be observed for named entities (NE): there are dimension reserved for different kind of abbreviations (German news organizations, big political parties, country codes), one with common male first names, and one for geographic locations. For the PIDAT tag, we see that often morphosyntactically similar words (in particular with the same ending) are grouped. Note that with a different initialization of the random seed we not only find the specializations swapped between the different latent dimensions but may find overall different patterns whereas the number of splits per POS tags tend to be rather stable. For instance, for random seed 3 we find that the PIS category (substituting indefinite pronoun) tends to group negative (nichts, niemand) and assertive existential (einer, eine, eines, einige) pronouns. With the other random seeds, we rather have patterns that indicate that a particular word can occur in a similar context.

7.8 External comparison and conclusions

Finally, we compare the best grammars that we trained with alternative approaches from the literature using the test sets of the different treebanks. In doing so we consider two kinds of comparisons:

1. How do state-refined hybrid grammars compare to other probabilistic grammar-based models?

2. How do state-refined hybrid grammars compare to different generations of discriminative parsers?

Experimental results for NeGra and TiGer are depicted in Table 7.21 and Table 7.22, respectively. The rates of sentences that could not be parsed are 5 (gold POS) of 1000 for NeGra, 112 (gold POS) / 128 (predicted POS) of 5047 for HN08, and 147 (gold POS) / 165 (predicted POS) of 5000 for SPMRL. This gives still some room for improvement by, e.g., improving the lexical layer, using a higher coverage child-labeling grammar if the strict-Markov grammar fails, or constructing unsound parses ad-hoc based on entries in the chart.

Addressing the first question we see that hybrid grammars tend to obtain state-of-the-art accuracy when compared to discontinuous grammar-based models in the strict

[22]The scores are based on a replication by Fernández-González and A. F. T. Martins (2015).

	Method	F1 ($\leq$ 40)	F1 (all)
this work			
hybrid-f2$_{\text{child}}$/max-rule-product		81.07	80.30
hybrid-f2$_{\text{child}}$/max-rule-product (product model)		82.51	81.70
Kallmeyer and Maier (2013) †	LCFRS	75.75	-
van Cranenburgh, Scha, and Bod (2016)	DOP	76.8	-
Maier (2015) †	SR-swap	76.95	77.0
Fernández-González and A. F. T. Martins (2015)	dep2const	81.08	80.52
Coavoux and Crabbé (2017)	SR-gap	82.76	82.2
Coavoux and Cohen (2019)♣💀	SR-set	-	**83.2**
Coavoux, Crabbé, and Cohen (2019)♣💀	SR-gap-unlex	-	**83.2**
Stanojević and Garrido Alhama (2017)💀	SA-swap	**83.39**	82.87

† sentence length $\leq$ 30, no discounting of root notes in F1-score
♣ predicted pos tags
💀 neural scoring component

Table 7.21: Results on the NeGra test set with gold POS tags.

sense (i.e., no pseudo-projective approaches). The split/merge algorithm significantly improves the baseline grammars while maintaining coverage and outperforms unrefined LCFRS from the literature and discontinuous tree substitution grammars obtained according to the DOP principle on several data sets. However, the pseudo-projective PCFG-LA-based approach by Versley (2016) outperforms our split/merge-refined hybrid grammars. Versley (2016) uses a linguistically motivated (de)projectivization strategy (see Section 8.4) that seems to address the sparsity of discontinuous constituents in the data very well. An explanation for the success of the pseudo-projective approach might be that discontinuous trees that actually occurs in treebanks can mostly be projectivized and deprojectivized without losses (Boyd 2007). Meanwhile, strictly discontinuous grammar formalisms that make available a large number of discontinuous productions (based on scarce evidence) do rarely benefit from the additional expressiveness.

Concerning the second question, we note that split/merge-refined grammars outperform early versions of transition-based discontinuous constituency parsing as proposed by Versley (2014a), Maier (2015), and Maier and Lichte (2016) and the dependency conversion-based parser of Hall and Nivre (2008). On the smaller NeGra corpus hybrid grammars also outperform the approach by Fernández-González and A. F. T. Martins (2015), that depends on a dependency-to-constituency transformation. These models constituted the state of the art in discontinuous constituent models when we started the development of the generalized split/merge procedure in 2016.

Clearly, the transition-based models developed by Coavoux and Crabbé (2017), Corro, Le Roux, and Lacroix (2017), and recent models that utilize neural nets are

	Method	SPMRL		HN08$^{\text{test}}_{\leq 40}$	
		F1 `spmrl`	`proper`	F1	EM
gold POS tags					
Gebhardt (2018) LCFRS_{ho} no prior/max-rule-prod.		75.00	75.08	79.29	42.55
Gebhardt (2018) hybrid-f2_{child} no prior/max-rule-prod.		72.91	72.98	77.68	41.28
hybrid-$\text{f2}_{\text{strict-Markov}}$ prior 2.0/max-rule-prod.		76.58	76.66	80.74	43.80
hybrid-$\text{f2}_{\text{strict-Markov}}$ prior 2.0/max-rule-prod. (product)		77.63	77.72	81.60	45.23
Hall and Nivre (2008)[22]	dep2const	-	-	79.93	37.78
Fernández-González and A. F. T. Martins (2015)	dep2const	80.62	-	**85.53**	**51.21**
Corro, Le Roux, and Lacroix (2017)	dep2const	-	**81.63**	-	-
Versley (2014a) and Versley (2014b)	EasyFirst-swap	76.11	-	74.23	37.32
Maier (2015)	SR-swap	-	74.7	79.52	44.32
Maier and Lichte (2016)	SR-swap	-	76.46	80.02	45.11
Coavoux and Crabbé (2017)	SR-gap	**81.50**	**81.60**	85.11	-
Stanojević and Garrido Alhama (2017)	SR-adj-swap	-	**81.64**	85.25	-
predicted POS tags					
Gebhardt (2018) LCFRS_{ho} no prior/max-rule-prod.			-	76.91	39.22
Gebhardt (2018) hybrid-f2_{child} no prior/max-rule-prod.			-	75.66	38.40
hybrid-$\text{f2}_{\text{strict-Markov}}$ prior 2.0/max-rule-prod.			73.86	78.38	40.59
hybrid-$\text{f2}_{\text{strict-Markov}}$ prior 2.0/max-rule-prod. (product)			74.80	79.22	41.86
Versley (2016)	pseudo-proj. PCFG-LA (product)		79.50	82.93	44.26
van Cranenburgh, Scha, and Bod (2016)	DOP		-	78.2	40.0
Coavoux and Cohen (2019)	SR-set		82.5	-	-
Coavoux, Crabbé, and Cohen (2019)	SR-gap-unlex		**82.7**	-	-

Table 7.22: Results on the TiGer test sets.

significantly more accurate than the grammars proposed in this thesis. This trend resembles the one for continuous parsing: the Berkeley parser, which for some time provided state of the art results in various languages, is outperformed by discriminative systems (L. Huang 2008b). This holds in particular, if the parsers utilize neural nets (Durrett and Klein 2015; Shen et al. 2018; Kitaev and Klein 2018).

What are the possible reasons for these differences? On the one hand there are restrictions in consequence of using a grammar-based approach. Clearly, the coverage of the development set and the number of sentences for which our grammars cannot provide any parse, are a limiting factor, when comparing to the transition systems by Versley (2014a), Maier (2015), and Coavoux and Crabbé (2017), which in principle can generate any discontinuous structure. However, our oracle experiments indicate that there is still potential that we do not leverage due to the state behaviour and the weight structure of our grammar.

A detailed understanding of the limitations of the latter kind and why discriminative (in particular neural) models improve over grammar-based models is beyond the scope of this thesis, but we want to pin down three possible causes of the problem.

a) We conjecture that the context-free independence assumptions epitomized by probabilistic RTG in the backbone of IRTG are too much a restriction. Our experiments as well as earlier work (on parsing and machine translation) where discriminative (neural) models outperform the probabilistic grammar-based ones support this assumption (Durrett and Klein 2015; Sennrich, Haddow, and Birch 2016).

b) A more conservative explanation is that grammar-based models are in principle sufficiently expressive but that there are inherent limits to the quality of a model that we can obtain by split/merge training. This hypothesis is supported by our observation that initial nonterminal granularity seems to have a higher influence on the grammar quality than one might would expect: if a good grammar cannot be realistically obtained from a baseline that is too coarse, then split/merge training might be insufficient to result close to the best possible grammar refinements and weight assignments without external guidance. Recall also our supposition that certain long-range dependencies might be hard encoded into nonterminals from automatic extraction that were not retrieved automatically. This is especially supported by our experiment where the oracle reranking scores are above the state-of-the-art, which indicates that the full potential is not realized and better weight-structures may be obtained.

c) Another obstacle might be approximative parsing objectives that we used. Unfortunately, due to the NP-hardness of exact PRTG parsing a clarifying experiment seems to be infeasible. Even if we solved this problem it would still be open how such an algorithm can be used to resolve ambiguity of different operator trees that are interpreted to the same parse tree.

8 Comparison with related work

The problem of data-driven syntactic parsing and that of discontinuous and non-projective parsing, in particular, has been intensively studied in the last three decades. We review some this work in this section. Before we explain specific models in more detail, we give a brief overview of different lines of research.

Discontinuous/non-projective corpora. A first precondition for the creation of data-driven parsers for non-projective or discontinuous parsing is the availability of corpora with discontinuous or non-projective syntactic structures. For German, the NeGra (Skut et al. 1997) and the TiGer (Brants et al. 2004) corpora have been created. Together with NeGra, a data format called *NeGra export format* has been proposed, which is capable of representing discontinuous constituent structures. The format also supports so-called secondary edges for ambiguous or coordinated relations between words and phrases.

The discontinuous Dutch Lassy corpus (van Noord 2009) has been released in 2009. Also, the Penn Treebank (M. P. Marcus, Santorini, and Marcinkiewicz 1993) contains so-called trace nodes, which encode that arguments have been moved to different locations in the parse tree. Usually, these trace nodes are removed to obtain continuous trees recognizable by CFG. However, Evang and Kallmeyer (2011) also derived a discontinuous version of the Penn treebank from the trace nodes.

Treebanks containing non-projective dependency trees can be obtained by converting a discontinuous constituent treebank (Xia and Palmer 2001). The process is based on a hand-crafted set of rules that assigns a head to each constituent. In the constructed dependency tree, this head dominates the heads of the child constituents. A conversion of TiGer has, for instance, been used in the CoNLL-X shared task on dependency parsing (Buchholz and Marsi 2006). Alternatively, there are native dependency treebanks, such as the Prague dependency treebank (Hajič et al. 2000), which feature non-projective dependency structures. Many of the recent developments in the creation of dependency corpus resources are connected to the *Universal Dependency* project (Zeman et al. 2019), which promotes a cross-lingual annotation scheme.

Non-projective dependency parsing. Due to the high computational costs of parsing with mildly context-sensitive grammars when compared to CFG, most of

	year	parsing complexity
Grammar-based models (generative, probabilistic)		
LCFRS[1]	2010	Viterbi: $O(n^{3k})$[2]
discontinuous TSG (DOP)[3]	2012	Viterbi: $O(n^{3k})$[2]
LCFRS/sDCP hybrid grammar[4]	2014	Viterbi: $O(n^{3k})$[5]
Transition-systems (discriminative)		
EasyFirst-swap[6]	2014	$O(n^3)$
ShiftReduce-swap[7]	2015	Greedy: $O(n^2)$
ShiftReduce-gap[8]	2017	Greedy: $O(n^2)$
ShiftAdjoin-swap [9]	2017	Greedy: $O(n^2)$
ShiftReduce-set[10]	2019	Greedy: $O(n)$[11]
Graph-based parsers (discriminative)		
dep2const[12]	2008	Viterbi/Greedy: $O(n^2)$[13]

Table 8.1: Categorization of implemented models for discontinuous constituent parsing. The year indicates the publication of first practical experiments with the model. The parsing complexity states the worst-case complexity of the parsing algorithm with which the models are usually applied given the sentence length n.

the early work on data-driven parsing focused on CFG (Chitrao and Grishman 1990; Charniak 1996; M. J. Collins 1999; Charniak 2000) and related formalisms (e.g., TSG, Bod 2003; Prescher et al. 2003). During the first decade of the 21st century, dependency parsing has gained in popularity (Nivre 2003; Buchholz and Marsi 2006). On the one hand, dependency trees contain the syntactic information that is sufficient for various downstream applications (Culotta and Sorensen 2004; Ding and Palmer 2005; Snow, Jurafsky, and Ng 2005). On the other hand, the considered models often have parsing algorithms that run in time linear or quadratic in the input sentence's length, which is much faster than the cubic complexity of CFG parsing. One approach uses the maximum spanning tree (MST) algorithm (McDonald et al. 2005), which supports non-projective structures directly. Also, transition systems that can produce (a subset of) non-projective dependency structures have been proposed (Nivre and Nilsson 2005; Nivre 2009).

Discontinuous constituent parsing. The study of parsers based on LCFRS for discontinuous phrase structure parsing started in 2008. Most work focuses on discontinuous constituent parsing, but there have been theoretical (Kuhlmann and Satta 2009; Kuhlmann 2013) and practical (Maier and Kallmeyer 2010) investigations into the induction of lexicalized LCFRS for dependency parsing as well. Progress was made during the following years, e.g., parsers became faster due to heuristics (Angelov and Ljunglöf 2014) and coarse-to-fine approaches (van Cranenburgh 2012; Ruprecht and Denkinger 2019). Also the accuracy could be improved by incorporating context-dependent category splits Kallmeyer and Maier (2013). Nevertheless, LCFRS-based discontinuous parsing mostly remained a niche topic. Presumably, the main reasons for this are that the run time of these LCFRS-based models is still orders of magnitudes higher than that of dependency parsers and that the accuracy remained lower than that of parsers for continuous or projective parsing. While all these LCFRS-based models are generative, there have been efforts to specify discriminative models for discontinuous constituent parsing too. Notably, conversion schemes from discontinuous constituent trees into non-projective dependency structures with rich edge labels have been introduced (Hall and Nivre 2008; Fernández-González and A. F. T. Martins 2015; Corro, Le Roux, and Lacroix 2017). In order to obtain a dependency tree, first a dependency parser is run (in linear or quadratic time) and then the conversion is undone. Later also transition-systems that allow direct creation of constituent structures have been invented (Versley 2014a; Maier 2015; Coavoux and Crabbé 2017; Stanojević and Garrido Alhama 2017; Coavoux and Cohen 2019; Coavoux, Crabbé, and Cohen 2019). An overview of different approaches to discontinuous constituent parsing and their time complexity is given in Table 8.1.

Mildly context-sensitive continuous parsing. Related to the research on discontinuous parsing is the investigation of so-called *mildly context-sensitive* (Joshi

[1] Vijay-Shanker, Weir, and Joshi (1987), Maier and Søgaard (2008), Kallmeyer and Maier (2010), and Kallmeyer and Maier (2013).
[2] For a binarized grammar with maximal fanout k.
[3] van Cranenburgh, Scha, and Sangati (2011) and van Cranenburgh, Scha, and Bod (2016).
[4] Nederhof and Vogler (2014) and Gebhardt, Nederhof, and Vogler (2017).
[5] For a binarized grammar with maximal fanout k in string component.
[6] Versley (2014a).
[7] Maier (2015) and Maier and Lichte (2016).
[8] Coavoux and Crabbé (2017) and Coavoux, Crabbé, and Cohen (2019).
[9] Stanojević and Garrido Alhama (2017).
[10] Coavoux and Cohen (2019).
[11] Although the length of each trajectory is linear in the sentence length n, the number of actions that needs to be scored can be in $O(n^2)$.
[12] Hall and Nivre (2008), Versley (2014a), Fernández-González and A. F. T. Martins (2015), and Corro, Le Roux, and Lacroix (2017).
[13] The complexity depends on the used dependency parser and, in case of graph-based parsing, the involved features.

1985) grammar formalisms, i.e., formalisms that extend context-free grammar, that can represent a limited number of crossing dependencies, have the *constant growth* property, and that are parsable in polynomial time. Joshi (1985) argues that grammars with these properties are likely to be adequate to model natural language: the first two properties are needed to address certain phenomena, whereas the last two properties restrict the grammars to allow feasible computation. While LCFRS is mildly context-sensitive, many of the formalisms considered generate (at first sight) continuous analyses.

A prominent member of this class is *tree adjoining grammar* (TAG, Joshi, L. S. Levy, and Takahashi 1975), which is a tree-generating device that allows for a very restricted form of second-order substitution. Specifically, the class of TAGs is equivalent to that of linear and monadic context-free tree grammars (Kepser and Rogers 2011; Gebhardt and Osterholzer 2015). TAG is sufficient to model various linguistic phenomena and broad coverage grammars have been hand-developed for languages such as English and French (see Kallmeyer 2010, for an overview). Still, there are phenomena such as scrambling that occur in free-word order languages but cannot be adequately modeled with TAG. Extensions of TAG such as v-TAG (Rambow 1994) and *tree tuple multi-component TAG* (TT-MCTAG, Lichte 2007) have been proposed to this end. Although the trees generated by TAG are continuous, there are approaches that rewrite the TAG derivations to non-projective dependency structures (Kallmeyer and Kuhlmann 2012).

Combinatory categorial grammars (CCG, Ades and Steedman 1982; Steedman 1987; Szabolcsi 1989) are weakly equivalent to TAG (Vijay-Shanker and Weir 1994) and attracted a lot of interest over the last decades (Hockenmaier and Steedman 2002; Clark, Hockenmaier, and Steedman 2002; Clark and Curran 2007; Lewis and Steedman 2014). In contrast to the grammar formalisms considered so far, CCG is not based on a rewriting mechanism but on combinatorial logic: first, a primitive type (e.g., S or N) or a complex type (e.g., S/N) is assigned to each lexical entry. Afterward, types can then be combined according to a set of abstract combination rules (e.g., S/N and N may be combined to S). As a consequence, the design of the lexicon and the selection of relevant types play a central role. The latter has been successfully targeted with *supertagging* approaches (i.e., the search space during parsing is pruned by restricting the possible types for a given lexical utterance, see Clark and Curran 2004; Lewis and Steedman 2014). There are variants of CCG that use different sets of combination rules, which influences the generative capacity and parsing complexity. We refer to Kuhlmann, Koller, and Satta (2015) for an overview.

Other linguistically motivated expressive grammar formalisms are *headed phrase structure grammar* (HPSG, Pollard and Sag 1994) and *Linear functional grammar* (LFG, Kaplan and Bresnan 1982; Bresnan 2001). Both formalisms are constraint-based, i.e., the constituents are equipped with features (e.g., morphological features) and the grammar rules are extended by constraints on these features. Similar to CCG, the lexicon plays an important role as it provides possible POS tags and

features for each lexical entry. LFG and HPSG grammars have been hand-written for English (Flickinger 2000), German (Müller and Kasper 2000), and French (de Alencar 2017), *inter alia*, however, it is also possible to extract grammars automatically from treebanks after applying appropriate transformations (Cahill et al. 2002; Nakanishi, Miyao, and Tsujii 2004; Rehbein and van Genabith 2009). We refer to Müller (2016) for an overview of the above mentioned and other grammatical theories. In the following comparison, we only consider formalisms that can produce discontinuous analyses in a strict sense. In particular, we only consider data-driven models that reproduce the discontinuous or non-projective syntactic analysis as found in some treebank.

8.1 Transition-based dependency and constituent parsing

Transition systems play an important role in modern syntactic parsing (Nivre 2008). Formally, a *transition system* is a tuple comprising a set of configurations, a finite set of actions, an initialization function, a set of final configurations, and a finalization function. The initialization function maps each input into some initial configuration. Each action is a partial function that maps some admissible configuration into a successor configuration. Actions are applied repeatedly until a final configuration is obtained. Then, using the finalization function, the resulting parse is extracted from the final configuration.

A wide range of transition systems has been proposed that mainly differ in how the configurations and actions are defined (cf. Nivre 2008). For instance, for dependency parsing, each configuration often consists of a stack, a buffer, and a set of arcs. Initially, the stack contains some root token, the buffer contains the input sentence, and the set of arcs is empty. Actions either shift tokens from the buffer to the stack or create labeled arcs between the top of the stack and the first element of the buffer (while removing the dominated token). Once just the root remains, the set of arcs represents the dependency tree. In order to obtain non-projective edges, one *raises* crossing edges (Nivre and Nilsson 2005), which is remembered in the edge labels, until a projective tree is obtained. After projective parsing, the raising is undone during the finalization. Alternatively, one adds a mechanism that changes the order of the items in the stack or the buffer, or allows for connecting non-adjacent items (Attardi 2006; Nivre 2009).

Crucially for the success of a transition system is an *oracle* that selects the action that is applied. Often discriminative classifiers are used as an approximation of such an oracle. The classifier weighs the actions given the current configuration (Hall, Nivre, and Nilsson 2006). Either the highest weighted (and applicable) action is chosen greedily, or a fixed-sized beam with the best configurations is maintained (Y. Zhang and Clark 2011), where the weight of a configuration is the sum of the weights of the actions that were taken to reach it. The overhead of maintaining a beam is

linear in the size of the beam. Clearly, beam-search is prone to search error, i.e., the highest-scoring configuration is not reached because one of its predecessors was pruned from the beam. An effective mitigation is global normalization of action weights (Andor et al. 2016). Kuhlmann, Gómez-Rodríguez, and Satta (2011) developed a dynamic programming algorithm that is capable of computing the globally optimal transition for certain transition systems depending on the used features.

In order to learn the classifier, the training data is converted to so-called gold transitions that reconstruct the correct parse tree from the input sentence. Next, pairs of configurations and gold actions are given to some off-the-shelf machine learning algorithm, which can be based on support vector machines, linear models over manual-designed features, or neural networks. Another option is repeated querying of the parser and updating weights according to the perceptron algorithm. It was found that only exploring the gold transitions yields to degenerate parsing results once the parser leaves the gold transitions at test time or if there are multiple correct actions. Hence, a line of work initiated by Goldberg and Nivre (2013) explores so-called dynamic oracles that compute, given any configuration, the optimal action. Here, an optimal action is the one from which the best reachable final configuration is still reachable. The final configurations are compared with the correct parse tree and a score is computed according to some metric. Dynamic oracles are not known for every transition system and sometimes too costly to compute. Alternatively, reinforcement learning techniques, that explore suboptimal derivations at random, have been investigated (Lê and Fokkens 2017; Fried and Klein 2018).

In the following, we review some transition systems that have been proposed for non-projective dependency parsing and discontinuous constituent parsing.

8.1.1 Pseudo-projective dependency parsing

Pseudo-projective dependency parsing is grounded on a raising (or lifting) operation proposed by Kunze (1968) and Kahane, Nasr, and Rambow (1998): a non-projective dependency tree can be made projective by attaching each child node c that leads to a violation of the projectivity condition for its parent n to some ancestor node a that also spans the gap of n. Clearly, such a node exists because the root covers all sentence positions.

Nivre and Nilsson (2005) developed a non-projective dependency parser based on this idea. They transform a non-projective dependency corpus in a projective one but encode the raising in the edge labels of the dependency tree. Then, they train a transition system for projective dependency parsing on this corpus. Finally, the non-projective dependency tree is restored by undoing the raising. The accuracy with which this reverse transformation can be performed depends on the detail of information in edge labels: due to sparsity problems, the performed raising cannot be encoded precisely. Different granularities of edge labels are compared empirically. Since the algorithms for greedy projective dependency parsing have linear complexity

in the length n of the input, the parsing complexity for this approach is $O(n)$.

8.1.2 Swap transitions for dependency and constituent parsing

Non-projective dependency structures can be directly obtained with a transition system if it features a SWAP action (Nivre 2009; de Lhoneux, Stymne, and Nivre 2017). This allows for free rearrangement of the input tokens and, in consequence, the creation of crossing arcs.

A similar idea was applied to constituent parsing by Maier (2015). He extends a transition system for continuous phrase structure parsing by Zhu et al. (2013), defined as follows. Configurations consist of a stack and a buffer containing (partial) trees. Initially, the words of the sentence are in order in the buffer. Tokens can be shifted from the buffer to the stack, POS-tags can be assigned to the top-most word, and the two top-most trees on the stack can be extended by a constituent to form a new tree. These actions allow for the creation of arbitrary continuous, binary trees. If the second token on the stack is shifted back to the buffer, then arbitrary discontinuous, binary trees can be obtained.

The *easy-first* transition system (Goldberg and Elhadad 2010) follows a different paradigm to build parse trees. Instead of reading the sentence from left to right, a list of partial trees is iteratively updated. Initially, the list contains the input words in order. Adjacent tokens in the list may be combined, where the *easiest* combination, i.e., the highest scoring one, is performed *first*. Again, non-projective or discontinuous structures may be obtained by swapping tokens in this list (Versley 2014a; Versley 2014b).

Stanojević and Garrido Alhama (2017) include a swap action in yet another transition system that features a sister-adjunction operation (Cross and L. Huang 2016). Using a neural network-based discriminative classifier and a training methodology that favors delayed (lazy) swapping, they are able to predict discontinuous structures with competitive accuracy.

One obstacle in applying these systems is that long-range dependencies within a discontinuous or non-projective tree may require long sequences of SHIFT and SWAP actions similar to ordering with the *bubble sort* algorithm. This makes it difficult for an oracle, that is queried before each action, to predict them correctly. Maier (2015) tries to remedy this problem with a compound swap action that swaps multiple items in one step to reduce the number of actions and obtains a higher accuracy. Alternatively, Maier and Lichte (2016) introduce a SKIPSHIFT-i action that shifts the i-th item from the buffer to the stack. The worst-case complexity for greedy parsing with transition systems extended by a SWAP action is, similar as for bubble sort, $O(n^2)$. However, empirically SWAP actions are not triggered very often, leading to expected linear parsing complexity (Nivre 2009).

8.1.3 Attardi's system

Non-projective dependency structures can also be obtained with the transition system by Attardi (2006). It adds six kinds of transitions to the arc standard system for projective dependency parsing. LEFT2, LEFT3, RIGHT2, and RIGHT3 create left or right arcs between the top-most element of the buffer and the second or third element of the stack. This is sufficient to create many of the non-projective structures that occur in the Prague dependency treebank. For the remaining cases, there is an EXTRACT action, which moves the second item from the stack on an additional temporary stack, and an INSERT action, which moves the item back to the primary stack. A restricted version of Attardi's system is investigated by Kuhlmann and Nivre (2010). Only the LEFT2 and RIGHT2 actions are kept, which restricts the non-projectivity that can be obtained. Later Gómez-Rodríguez, Sartorio, and Satta (2014) present a dynamic oracle for this restriction.

8.1.4 Covington parser (SR-GAP) for discontinuous constituent parsing

An alternative way to obtain discontinuous or non-projective structures was proposed by Covington (2001) and empirically explored by Marinov (2007). The transition system is classified as *list-based* by Nivre (2008) and also called *Covington parser*. Its configurations consist of a stack with an additional pointer, a buffer, and, at least for dependency parsing, a set of arcs.[14] The buffer is initialized with the input sentence. The computations follow a particular order:

1. A token is shifted from the buffer to the top of the stack. The additional pointer is located at the second element of the stack.

2. The stack pointer may be moved down the stack by one (an action called GAP) for an arbitrary number of times.

3. The top-most stack item and the item under the stack pointer are combined (*reduced*), and the stack pointer is reset to the second item on the stack.

4. Steps 2 and 3 are repeated until the stack pointer reaches the bottom of the stack, or a new item is shifted.

This way, any non-projective (or binarized discontinuous) structure can be obtained. The variant for discontinuous constituent parsing for this algorithm was introduced by Coavoux and Crabbé (2017) and later refined by Coavoux, Crabbé, and Cohen (2019).

Similar to repeated SWAP actions, there is the problem that the oracle needs to predict multiple GAP actions before the reduce action can be performed. To

[14] Nivre (2008) gives a technically different characterization based on two lists.

this end, Coavoux and Crabbé (2017) also consider a MultiGap action, which performs multiple Gap actions in an atomic step. Recently, Coavoux and Cohen (2019) found a different solution by generalizing the "stack with pointer" architecture to a random-access set. Here, a set replaced the stack. Next to the usual buffer, there is a distinguished item called *focus item*, which may be reduced with any item of the random-access set. The highest scoring reduction is performed and yields the new focus item. This process is repeated until it is more likely to shift a new word from the buffer to become the next focus item. Using neural network-based classifiers, Coavoux and Cohen (2019) obtain state-of-the-art performance with this transition system.

8.2 Graph-based parsing algorithms

A second major paradigm in the modern parsing literature is based on finding maximum spanning trees (MST) in labeled fully connected graphs (McDonald et al. 2005). For a given sentence, a graph is constructed as follows. Each input token corresponds to a node in the graph. Directed labeled edges represent possible attachments. This way, non-projective structures are supported off the shelf because no projectivity constraints are enforced when searching the maximum spanning tree. If the weight of the edge is independent of the other edges in the spanning tree, then the maximum spanning tree can be found in quadratic time in the sentence length with the algorithm by Chu and Liu (1965) and Edmonds (1967). With J. M. Eisner's algorithm (1996), which has cubic runtime in the sentence length, it is possible to restrict the search space to just projective trees.

In order to assign the weights to the edges, different machine learning models such as linear models (McDonald et al. 2005), support vector machines (Corston-Oliver and Aue 2006), and neural networks (Kiperwasser and Goldberg 2016) are used. The restriction of computing the edge weights independently of the other edges in the spanning tree was found to be a limiting factor. To this end, higher-order features that consider sibling and grand-parent relations in the spanning tree have been investigated at the expense of losing optimality guarantees (Koo et al. 2010; H. Zhang and McDonald 2012).

The graph-based method has also been adapted for constituent parsing. To this end, the constituent tree is cut into linear spines. Each of these spines covers exactly one word of the sentence. If each spine is adjoined into a particular other spine, then the original constituent tree is recovered. Hence, the constituent tree can be represented as a dependency graph, where each vertex corresponds to a word, the parent of each word determines the adjunction site, and the edge label the nodes on the spine. For parsing, a fully connected graph, that encodes different adjunction and spine options, is considered. The best edges are again selected with the MST algorithm or some approximation in case of higher-order features. This view was

first explored for continuous phrase structure parsing by Carreras, M. Collins, and Koo (2008). Hall and Nivre (2008), Fernández-González and A. F. T. Martins (2015), and Corro, Le Roux, and Lacroix (2017) apply similar strategies in a discontinuous setting, which mainly differ in the information that is encoded into the edge labels.

8.3 Reduction to sequence-to-sequence and sequence labeling tasks

Due to the increasing performance neural models in recent years (see Goldberg 2017, for an overview), it has been proposed to reduce parsing to sequence to sequence (SEQ2SEQ) and sequence labeling tasks. A SEQ2SEQ model reads in a sequence of tokens and outputs another sequence of tokens (of potentially differing length). During sequence labeling, for each input token exactly one output label is determined. With a neural network, a real-valued vector is computed as a *representation* for a particular word or a sequence of words. To this end, different architectures such as *long-short term memory* (LSTM, Hochreiter and Schmidhuber 1997), *gated recurrent units* (GRU, Cho et al. 2014), and the transformer architecture (Vaswani et al. 2017) are used. Pretraining these models on large amounts of unlabeled data, i.e., large chunks of text crawled from the web, Wikipedia, or books, is an important aspect when building these models. Using the computed representation of a word or a sequence of words, either a distribution over different output labels is computed (sequence labeling) or the next output action is predicted (SEQ2SEQ).

In order to reduce constituent parsing to a sequence labeling or a SEQ2SEQ task, one needs to encode which constituent spans which words in a suitable, compact way. As with SEQ2SEQ the length of the output is unrestricted, Vinyals et al. (2015) split up the bracketed term (that represents the parse tree) into tokens. For instance, the tree '(NP (DET NN))', which could be the preferred parse for "the house", is decomposed into a sequence of length four: '(NP', '(DET', 'NN', ')$_{\text{NP}}$'. This approach is currently only applicable to continuous constituent trees. It achieves competitive performance if parse trees for unlabeled sentences derived by some baseline parser are added to the training set.

In case of sequence labeling, the length of the output sequence needs to match the length of the input. Several encodings for continuous constituent trees have been proposed, all of which assume binary trees without monadic rules. Gómez-Rodríguez and Vilares (2018) discuss different variants to represent a tree by storing for each pair of adjacent words the number of common ancestors. This encoding requires an infinite label set if all possible trees shall be represented.

A compact encoding is due to Kitaev and Klein (2019). It exploits the fact that the number of unlabeled binary trees for a sentence of length n is bounded by the n-th Catalan number (which is bounded by $4n$). Each of these trees has $n-1$ internal nodes and n leaves. The tree is represented by deciding for each node if it is the left

or right child of its parent. Hence, for each sentence position one has to make two binary decisions. For labeled parsing, these decisions can be augmented with a label for the internal node. This leads to a finite label set of size $4 \cdot |N|$, where N is the set of nonterminal labels. As not every sequence of decisions encodes a well-formed binary tree, Kitaev and Klein (2019) give a dynamic programming solution that selects the best well-formed parse in $O(n^3)$.

In principle, one strategy to encode dependency trees has been proposed, which also supports non-projective trees. Basically, for each sentence position the parent needs to be assigned, which Li et al. (2018) encode by an integer k. If $k > 0$, then the parent is the k-th token to the right. If $k < 0$, then the parent is the k-th token to the left. This encoding requires an infinite label set if arbitrary dependency trees shall be represented. Some variations in the encoding to weaken this problem are discussed by Strzyz, Vilares, and Gómez-Rodríguez (2019).

8.4 Pseudo-projective grammar-based approaches.

The approach of pseudo-projectivity has been applied in grammar-based parsing settings too (Johnson 2002; Dienes and Dubey 2003; R. Levy and Manning 2004; Schmid 2006; Cai, Chiang, and Goldberg 2011; Versley 2016); usually with the aim of constituent parsing. These models either work by node raising (i.e., discontinuous constituents are split, and some of their arguments are shifted to a position higher in the tree such that the tree becomes projective) or by splitting of nonterminals (see Boyd 2007 and van Cranenburgh, Scha, and Bod 2016). Hsu (2010) analyses how accurately a PCFG-based parser can predict the projectivized trees depending on the different conversion schemes. She finds that node-raising leads to more accurate predictions. However, this experiment does not take into account a back-transformation of the projectivized tree after parsing and measurement of F1 on the predicted discontinuous trees. Boyd 2007 introduced the splitting approach (each discontinuous node A is split up into a sequence of continuous nodes of the form A^*) because it is particularly easy to undo. While, in principle, the deprojectivization mapping can be ambiguous, she finds that this problem does not occur in practice if one undoes the projectivizations of discontinuous trees in TiGer. In her experiments, also the attachment accuracy is higher than with node-raising. We already mentioned an alternative approach to node splitting in the context of approximate LCFRS parsing. Here, we split up A to $A^*_1, \ldots, A^*_k$. However, note that due to the free word order (of German), different occurrences of A^*_i in the data may realize very unrelated aspects. Hence, we should not assume that parsers predict A^*_i correctly.

Versley (2016) successfully employs an approach that combines node-raising with the insertion of an additional nonterminal that encodes the type of the original parent and if this origin is to the left or to the right. He experiments with different strategies for deprojectivization and finds a bottom-up approach to work favorable. Overall, he

trains a PCFG-LA using these corpus transformations that can predict discontinuous trees very well.

Van Cranenburgh, Scha, and Bod (2016, Section 7) investigate, *inter alia*, an approach that combines pseudo-projective parsing with data-oriented parsing (see Section 8.5.1). Instead of using the LCFRS for providing candidate trees that are reranked according to the discontinuous DOP model, a PCFG and a continuous DOP model are used. Both models are induced from the projectivized treebank. Each nonterminal A_i^*, which simulates a discontinuous nonterminal A, still has an annotation that marks the component of A that is approximated. The best derivation according to the continuous DOP model is computed, and the split nonterminals are merged again to obtain a discontinuous tree. Surprisingly, the resulting parses are more accurate than those of the discontinuous DOP model.

Arguably, hybrid grammars, in particular those with fanout 1, can be framed as a pseudo-projective parsing mechanism: the expressiveness of the string component can be regular or context-free, yet non-projective structures may be obtained due to the tree component of the grammar. Here, the synchronous process can be seen as a transduction or post-processing step that rewrites the projective parse tree of the regular or context-free string grammar into a discontinuous one.

8.5 Related work on LCFRS parsing

Maier and Søgaard (2008) and Kuhlmann and Satta (2009) present ways to read-off LCFRS from discontinuous constituent trees and non-projective dependency trees, respectively. Later Maier and Kallmeyer (2010) present first experiments for data-driven parsing with LCFRS for either kind of syntactic annotation. Due to the long runtime of the LCFRS parsing algorithms, only sentences up to restricted length were considered (about 30). Since the algorithms for discontinuous dependency parsing based on transition systems and graph-based parsing are faster and more accurate, most of the subsequent research with LCFRS considers only constituent parsing.

One approach to tackle parsing complexity was the analysis of different binarization strategies. Recall that for each LCFRS where rules have more than two right-hand-side nonterminals, there is an equivalent LCFRS with no more than two right-hand-side nonterminals. The construction by Seki et al. (1991) does not preserve the fanout. In particular, how a given rule is decomposed into binary rules matters. Hence, one can look for an optimal binarization of a given grammar (Gómez-Rodríguez et al. 2009; Gómez-Rodríguez and Satta 2009; Kuhlmann and Satta 2009; Gildea 2010; Crescenzi et al. 2011) or a good trade-off between rank and fanout (Sagot and Satta 2010). This process can be quite time-consuming, but the computation can be done offline (i.e., just once).

Kallmeyer and Maier (2013) present an extensive study of the influence of various parameters of grammar induction, such as Markovization and binarization strategies.

The overall takeaway is that the binarization strategy does not seem to matter too much in practice. Markovization parameters can be optimized on a development set to yield a good trade-off between accuracy and coverage.

An alternative approach to handle parsing complexity is due to Maier, Kaeshammer, and Kallmeyer (2012). It follows from the observation that on the one hand, most discontinuous structures have at most one gap, i.e., LCFRS with fanout two suffice to handle them adequately. Even more, to many discontinuous trees with more gaps, a linguistically motivated transformation can be applied, which reduces the fanout in most cases to two as well. This restricted class of LCFRS admits more efficient parsing. Maier, Kaeshammer, and Kallmeyer (2012) implement a parsing algorithm and run experiments with grammars obtained from the restricted and transformed treebanks, which indicate vast speed-ups without loss in accuracy.

A similar study of gap-degree (Holan et al. 1998) in the dependency case is due to Kuhlmann (2013). Additionally, he considers a property called *well-nestedness* (Bodirsky, Kuhlmann, and Möhl 2005), i.e., the string regions spanned by two disjoint discontinuous subtrees do not interleave. He finds that most trees of dependency treebanks are well-nested and have a low fanout. A parsing algorithm for well-nested LCFRS with fanout k has a favorable parsing complexity of $O(|G| \cdot |w|^{2k+2})$. For LCFRS with unrestricted fanout and rank, Kuhlmann (2013) shows that solving the *universal recognition problem* (also known as *uniform membership problem*, i.e., the question if a word is generated by a grammar where the grammar is part of the input) for LCFRS is NP-complete. See Björklund, Berglund, and Ericson (2016) for a survey on the complexity of the (non-)uniform membership problem for other mildly context-sensitive formalisms.

Finally, a range of low-level optimizations has been applied to the LCFRS parsers in order to allow for more efficient parsing. The CKY-style chart-parsing algorithm by Seki et al. (1991) has been reconsidered in Burden and Ljunglöf (2005) and stated according to the *parsing as deduction* principle (Pereira and D. H. D. Warren 1983). Burden and Ljunglöf (2005) also derive additional parsing strategies such as incremental and approximate parsing. Notable implementations of LCFRS parsers are the grammatical framework (Ranta 2011) and `rparse` (Kallmeyer and Maier 2013). Angelov and Ljunglöf (2014) present a parser that uses a heuristic-driven incremental parsing strategy to reduce the search space and outperforms `rparse` by a large margin. In consequence, parsing of sentences up to length 40 seems feasible. The coarse-to-fine parser by van Cranenburgh (2012) (which uses a PCFG approximation of the probabilistic LCFRS) even allows for parsing with unrestricted length if suitable pruning parameters are used. Another feasible coarse-to-fine parsing approach based on a Chomsky-Schützenberger-representation of the LCFRS is due to Ruprecht and Denkinger (2019).

8.5.1 Discontinuous data-oriented parsing

Tree substitution grammars (TSG, Joshi and Schabes 1992) make use of a set of tree fragments that can be combined to form parse trees. A combination of two fragments is viable if the root of the first fragment equals one of the leaves of the second fragment. In this case, a larger tree is formed that is obtained by merging both fragments at the said root and leave node. The process of combining tree fragments to a terminal tree, i.e., one with a designated root symbol and just terminals as leaves, is called derivation. As usual, fragments with the same root node get assigned probabilities to obtain a probabilistic (or stochastic) TSG.

A terminal tree might be derived in different ways. Hence, its probability is obtained by summing over the probabilities of all these derivations. The parsing problem of TSG amounts to finding the tree with the highest probability mass for a given sentence. Because this problem is NP-hard (Sima'an 2002), only approximate solutions are calculated. One is by working with a reduction of TSG to CFG (Goodman 2003), where fragments are decomposed to smaller ones of height at most 2, which can then be interpreted as rules of a PCFG. After using a PCFG parsing pass, only the best candidates are reranked according to the TSG model.

Van Cranenburgh, Scha, and Sangati (2011) and van Cranenburgh, Scha, and Bod (2016) generalize TSG to discontinuous tree substitution grammars by allowing tree fragments to contain discontinuous nonterminal nodes. Similarly to the continuous case, there is a natural reduction of discontinuous TSG to LCFRS. We are particularly looking at discontinuous TSG with tree fragments obtained to the data-oriented parsing principle. *Data-oriented parsing* (DOP) was introduced by Bod (1992) for continuous constituent parsing and addresses the question of how tree fragments are obtained from a corpus. Essential considerations in this process involve the statistical stability of the induction strategy and the efficiency of the inference. Extracting every possible fragment usually slows down inference too much. A successful approach called *Double-DOP* (Sangati and Zuidema 2011) is using only fragments that occur at least twice in the training data.

During the reduction of TSG to CFG (or discontinuous TSG to LCFRS), the fragments of height two are augmented with substates. These substates can be interpreted as latent annotations and, thus, discontinuous TSG as latently annotated LCFRS. Hence, our split/merge-refined LCFRS are at least as expressive as the discontinuous DOP model by van Cranenburgh, Scha, and Bod (2016). When comparing the Double-DOP method with the split/merge-training algorithm, we note that Double-DOP is faster because it does not require numerous epochs of EM training and the computation of charts for the training sentences with respect to the grammar. The accuracy of our hybrid grammars seems to be en par or sometimes higher than the TSG of van Cranenburgh, Scha, and Bod (2016), however, in order to draw reliable conclusions, one should apply both methods to the same baseline grammar. Also, the influence of the parsing objectives should be controlled for.

8.5.2 Inclusion of function labels

Treebanks such as TiGer and NeGra also contain edge labels that indicate the grammatical function a child node has concerning its parent. This is necessary because the annotation scheme opts for flat hierarchies where other annotations encode at least some of these functions implicitly. However, also the Penn treebank, which favors a more deeply nested annotation scheme, uses 20 function tags. Hence, enriching the bare constituent structure by these grammatical functions is a relevant task. One option is to predict the function labels after the parse tree has been created. This approach has been followed by Blaheta and Charniak (2000). If a correct constituent parse is given, then they obtain an F1-score on function labels of 88.5. Alternatively, the edge labels can be jointly predicted with the constituent structure (Gabbard, Kulick, and M. Marcus 2006). Van Cranenburgh, Scha, and Bod (2016) follow this approach in the discontinuous setting by encoding the edge labels into the nonterminal names of the underlying LCFRS. Van Cranenburgh, Scha, and Bod (2016) obtain an F1-score of 93.5 on TiGer HN08 and 86.3 on the WSJ section of the Penn treebank if only the function labels of correct constituents of sentences of length ≤ 40 are considered.

Our implementation `panda-parser` allows for joint prediction of function tags as well. With hybrid grammars, these tags can be included in the sDCP component, i.e., function tags do not necessarily need to be encoded into the nonterminals. This circumvents data sparsity problems which might be connected to such an encoding. A detailed evaluation is left open for further research as well as the analysis if function labels should be included in nonterminal names. We note however that the hybrid grammar (product) in Table 7.22 in the predicted POS scenario obtains an F1-score of 93.2 on function tags for the TiGer $\text{HN08}_{\leq 40}$ test set, which is in the vicinity of van Cranenburgh, Scha, and Bod (2016).

8.6 Related work on grammars with latent annotations.

Following the proposal of PCFG with latent annotations by Matsuzaki, Miyao, and Tsujii (2005) and the success of the Berkeley Parser (Petrov et al. 2006; Petrov and Klein 2007) a range of variations of the PCFG-LA approach has been studied. Among others, the grammar formalism, the training process, and the inference algorithms have been investigated for both theoretical and practical aspects.

Ferraro, Van Durme, and Post (2012) consider an extension of the split/merge algorithm by a *coupling step*, which merges rules to tree fragments. This way, a tree substitution grammar with latent annotations is obtained that iteratively but gradually extends the number and size of the tree fragments. Although Ferraro, Van Durme, and Post just present a qualitative analysis, they find that this procedure yields linguistically plausible tree fragments and that split/merge refinement and additional expressiveness of TSG are complementary. This is in line with our observation that

the general structure and labeling scheme of the baseline grammar is relevant despite split/merge refinement.

In contrast to the split/merge method, Liang et al. (2007) and Liang, Jordan, and Klein (2010) investigate a version of refined PCFG-LA where the number of states splits depends on a hierarchical Dirichlet process that adheres certain prior parameters. The optimal number of splits is obtained via a variational inference scheme. Shindo et al. (2012) apply a similar idea to tree substitution grammars and obtain state-of-the-art results on the English Penn treebank.

Petrov (2010) shows that grammars trained from different random seeds exhibit a significant amount of variance. This variance can be exploited if product models, that combine grammars trained with different random seeds, are used. We could replicate these improvements with hybrid grammars.

Z. Huang and Harper (2009) note that after more than 6 split/merge cycles, the performance of PCFG-LA on English and Chinese data drops due to overfitting. They find it beneficial to include additional unlabeled data in the training process: the unlabeled data is parsed by the grammar, and in a second training run, the training data is extended by these *silver trees*. Z. Huang and Harper (2009) hypothesize that the additional noisy silver trees smooth the counts in the EM training favorably. This might be a better way for smoothing than with Dirichlet priors as we do in this thesis. However, we are not aware of such a comparison of self-training and priors for PCFG-LA. Z. Huang, Harper, and Petrov (2010) show that using product models and self-training in combination is complementary.

An alternative training methodology for PCFG-LA based on *spectral learning* is due to Cohen et al. (2012) and Cohen et al. (2014). Instead of the EM-algorithm, which only provides weight assignments that converge to a local optimum of the likelihood, a process based on singular value decomposition and dimensionality reduction is used to estimate the weights consistently. Cohen et al. (2013) present parsing experiments with PCFG-LA on the English Penn treebank, where the number of latent states is fixed in advanced and not adjusted according to the split/merge algorithm. In this setup, the spectral method slightly outperforms EM training in terms of accuracy and is considerably faster: the runtime of the spectral method is about the same as one epoch of EM training. Saluja, Dyer, and Cohen (2014) generalize this approach to single-nonterminal synchronous CFG (Chiang 2007) for machine translation and compare it to EM-based training too. They obtain significant improvements with spectral learning over a minimalistic baseline grammar with results comparable to the Hiero system (Chiang 2007).

Petrov and Klein (2008) present a discriminative version of their PCFG-LA framework. The resulting model is better described as a *conditional random field* (Lafferty, McCallum, and Pereira 2001), where each refined rule is a feature. The corresponding real-valued feature weights are trained with a gradient descent algorithm. Moreover, additional features (unknown word features and span features (Taskar et al. 2004)) can be added to the model. To make training and inference feasible, a coarse-to-fine

approach is used to compute approximate feature counts and prune entries in the fine charts. On the English Penn treebank (Wall Street Journal section), the discriminative version performs worse than the generative PCFG-LA model without the additional features but better in combination.

Dietze and Nederhof (2015) investigate a method for grammar induction that starts from the finest possible weighted tree automaton (that exactly generates the training data) and iteratively merges states under the restriction that the likelihood-loss is minimal and that the automaton resulting from the state-merge is still deterministic. Later Dietze (2018) presents an extended analysis of these methods as well as various theoretical properties of the split/merge procedure.

Concerning decoding, Petrov and Klein (2007) use a coarse-to-fine inference mechanism that involves pruning: instead of computing the full refined parse chart (that is in principle required by all the parsing objectives outlined in Section 3.4), one applies a pipeline with the increasingly refined grammars obtained after each of the split/merge cycles. The parse chart of the less-refined grammar serves as a filter for the more-refined one, where only cells with high probability are permitted to be filled. All weight-projections or Viterbi derivations can be computed on the pruned charts as well. Overall, hierarchical pruning is faster than computing the fine chart directly. The inexactness of pruning does not seem to be a problem in practice, which is not surprising given that all the objectives in use are approximations in the first place. Later, Yi et al. (2011) present a decoding algorithm that exploits the parallelism of modern GPU architectures. Although our pipeline uses coarse-to-fine parsing as well, it is of a different kind: the baseline probabilistic LCFRS is approximated by a PCFG to reduce the search space of the baseline LCFRS. Currently, our implementation, however, does not apply hierarchical pruning when solving the different parsing objectives and computes the refined chart for each entry in the baseline chart.

9 Conclusions

We conclude this thesis with a short summary of the results that we obtained and an outline of directions for future investigation. In the theoretic part, comprised of Chapters 2–6, we formalized a probabilistic variant of the IRTG framework. In particular, we presented a generalized version of the split/merge algorithm by Petrov et al. (2006) and discussed various parsing objectives for split/merge-refined grammars. That this generalization is feasible was not unexpected. Yet, the theoretical treatment indicates also gaps in the literature, e.g., the formulation of an exact parsing algorithm for IRTG.

Another formal contribution is the characterization of LCFRS/sDCP hybrid grammars as IRTG. In particular, we also provided a way to capture sDCP as IRTG. In contrast to Drewes, Gebhardt, and Vogler (2016), our method does not use a reduction to graph grammars but involves an additional alignment algebra. Due to the inherent complexity of sDCP, this approach is very specific but also technically involved. Hence, this characterization is probably less approachable than the rewriting or graph-based view and, thus, not the first choice for further formal investigation. We suppose, however, that alignment algebras between grammar formalisms that are simpler than sDCP (or where unranked hedges are replaced by ranked trees) are easier to handle.

We presented also a generalized induction method for LCFRS/sDCP hybrid grammars that employs decompositions instead of the recursive partitionings of Nederhof and Vogler (2014). We showed that the generalized version does allow to induce lexicalized hybrid grammars, however, we did not propose methods to obtain decompositions that guarantee lexicalization. Investigating such strategies could be subject of further investigation.

We presented constructions to obtain the parse charts for sDCP and LCFRS/sDCP hybrid grammars and argued that their computation has polynomial time and space complexity. These constructions are required to do EM training for a hybrid grammar given a corpus of hybrid trees and prepare the experiments.

In the applied part of this thesis, we showed that split/merge refinement generalizes to discontinuous approaches, mostly by experimentation on German treebanks. We confirmed many empirical results obtained for PCFG-LA also for LCFRS and hybrid grammars. Moreover, we explored parts of the hyperparameter space of the split/merge algorithm and found that the standard settings of the Berkeley parser

are not ideal for LCFRS and hybrid grammars. This closes a gap in the research landscape. To increase the confidence of these results, experimentation with topologically different languages and also dependency parsing are apparent next steps. There are some interesting aspects that are not yet understood when it comes to selecting appropriate baseline grammars for the refinement. In particular, the split/merge algorithm does not seem to recover all context-based information that we can hard-code into nonterminals during induction. Interestingly, with LCFRS we obtained worse results than with hybrid grammars – we do not yet know why this happens.

We improved the state-of-the-art of discontinuous grammar-based parsing in the strict sense. However, there is indication that strict discontinuous grammar-based parsing is worse than pseudo-projective parsing presented by Versley (2016). We have not yet excluded that this might be an artefact of differences in the implementation and the experimental setup. Thus, a one-to-one replication of the lexical smoothing mechanisms in the Berkeley Parser or a replication of the pseudo-projective method in our framework would be an interesting extension.

More general, despite the split/merge refinement, architectures without grammars tend to outperform grammar-based approaches by a significant margin. Here the trend of continuous parsing is mimicked. Hence, we must ask the question what motivates and legitimates further formal and practical investigation of grammar-based approaches. According to folklore knowledge, grammars are more interpretable than discriminative approaches, in particular neural ones. However, we see advances in the understanding of neural models in recent years (Linzen, Dupoux, and Goldberg 2016; Marvin and Linzen 2018; Goldberg 2019). It is also not plausible that our corpus-induced grammar with millions of rules is easy to interpret.

Recently, Nivre (2019) asked if the search for parsing architectures is still relevant if large general-purpose neural models, such as ELMO (Peters et al. 2018) and BERT (Devlin et al. 2019), can produce state-of-the-art results independent of a concrete architecture. For instance, Kulmizev et al. (2019) show that for dependency parsing the differences between transition-based and graph-based approaches have vanished. X. Zhang, Cheng, and Lapata (2017) present a simple algorithm for dependency parsing, which greedily selects the head for each word without enforcing the resulting graph to be a tree. Yet, with a neural scoring component they are able to produce high-quality well-formed dependency trees in 95% of the cases. Even more, for many downstream NLP applications syntactic features seem to be obsolete in light of models such as ELMO and BERT. This underlines that at least empirically we often do not need the complex grammars and algorithms that are interesting from the perspective of theoretical computer science.

Hence, where could be the competitive edge of grammar-based approaches and how could the performance gap be bridged? Concerning the latter, formal grammars have been combined with discriminative components, which prune the search space. This makes their application faster and more accurate (Vieira and J. Eisner 2017; Grünewald, Henning, and Koller 2018). Although we may apply the same technology

to systems based on hybrid grammars or LCFRS and are likely to obtain improvements in performance, it is not clear why one should use such a theoretically heavy model if simpler models do as well. Cases, where we anticipate formal grammars in the future, might be parsing tasks where "works very well in most cases" is not good enough, i.e., tasks where the parse tree shall meet qualitative requirements. One approach for semantic parsing in this direction is due to Groschwitz et al. (2018). These high-quality "type-checked" trees could be important for automated generation of data-base queries or the processing of legal or medical documents. However, such guarantees may only hold with hand-written grammars or very conservative data-driven ones.

Bibliography

Abney, Steven, David McAllester, and Fernando Pereira (1999). "Relating Probabilistic Grammars and Automata". In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*. ACL '99. College Park, Maryland: Association for Computational Linguistics, pp. 542–549. DOI: 10.3115/1034678.1034759.

Ades, Anthony E. and Mark Steedman (1982). "On the order of words". In: *Linguistics and Philosophy* 4 (4), pp. 517–558. ISSN: 1573-0549. DOI: 10.1007/BF00360804.

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins (2016). "Globally Normalized Transition-Based Neural Networks". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 2442–2452. DOI: 10.18653/v1/P16-1231.

Angelov, Krasimir and Peter Ljunglöf (2014). "Fast Statistical Parsing with Parallel Multiple Context-Free Grammars". In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 368–376. URL: https://www.aclweb.org/anthology/E14-1039.

Arnold, André and Max Dauchet (1975). "Transductions inversibles de forêts". Thèse 3ème cycle M. Dauchet. Université de Lille. URL: http://ori.univ-lille1.fr/notice/view/univ-lille1-ori-70098.

Attardi, Giuseppe (2006). "Experiments with a Multilanguage Non-Projective Dependency Parser". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City: Association for Computational Linguistics, pp. 166–170. URL: https://www.aclweb.org/anthology/W06-2922.

Baker, James K. (1979). "Trainable grammars for speech recognition". In: *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*. Ed. by D. H. Klatt and J. J. Wolf, pp. 547–550.

Bangalore, Srinivas and Aravind K. Joshi (1999). "Supertagging: An Approach to Almost Parsing". In: *Computational Linguistics* 25 (2), pp. 237–265. URL: https://www.aclweb.org/anthology/J99-2004.

Bar-Hillel, Yehoshua, Micha A. Perles, and Eli Shamir (1961). "On formal properties of simple phrase structure grammars". In: *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* 14, pp. 143–172.

Barthélemy, François, Pierre Boullier, Philippe Deschamp, and Éric Villemonte de la Clergerie (2001). "Guided Parsing of Range Concatenation Languages". In: *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France: Association for Computational Linguistics, pp. 42–49. DOI: 10.3115/1073012.1073019.

Baum, Leonard E., Ted Petrie, George Soules, and Norman Weiss (1970). "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". In: *The Annals of Mathematical Statistics* 41 (1), pp. 164–171. ISSN: 00034851. URL: http://www.jstor.org/stable/2239727.

Becker, Tilman, Michael Niv, and Owen Rambow (1992). *The Derivational Generative Power of Formal Systems or Scrambling is Beyond LCFRS*. Technical Report IRCS-92-38. Institute for Research in Cognitive Science, University of Pennsylvania.

Bender, Emily M. (2013). "Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax". In: *Synthesis Lectures on Human Language Technologies* 6 (3), pp. 1–184. DOI: 10.2200/S00493ED1V01Y201303HLT020.

Benesty, Jacob, M Mohan Sondhi, and Yiteng Huang (2007). *Springer handbook of speech processing*. Springer.

Berg-Kirkpatrick, Taylor, David Burkett, and Dan Klein (2012). "An Empirical Investigation of Statistical Significance in NLP". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 995–1005. URL: https://www.aclweb.org/anthology/D12-1091.

Bird, Steven and Edward Loper (2004). "NLTK: The Natural Language Toolkit". In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain: Association for Computational Linguistics, pp. 214–217. URL: https://www.aclweb.org/anthology/P04-3031.

Birkhoff, Garrett and John D. Lipson (1970). "Heterogeneous algebras". In: *Journal of Combinatorial Theory* 8 (1), pp. 115–133. ISSN: 0021-9800. DOI: 10.1016/S0021-9800(70)80014-X.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387310738.

Björkelund, Anders, Bernd Bohnet, Love Hafdell, and Pierre Nugues (2010). "A High-Performance Syntactic and Semantic Dependency Parser". In: *Coling 2010: Demonstrations*. Beijing, China, pp. 33–36. URL: https://www.aclweb.org/anthology/C10-3009.

Björklund, Henrik, Martin Berglund, and Petter Ericson (2016). "Uniform vs. Nonuniform Membership for Mildly Context-Sensitive Languages: A Brief Survey". In: *Algorithms* 9 (2). ISSN: 1999-4893. DOI: 10.3390/a9020032.

Björklund, Johanna, Frank Drewes, and Niklas Zechner (2015). "An Efficient Best-Trees Algorithm for Weighted Tree Automata over the Tropical Semiring". In: *Language and Automata Theory and Applications*. Ed. by Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe. Cham: Springer International Publishing, pp. 97–108. ISBN: 978-3-319-15579-1. DOI: 10.1007/978-3-319-15579-1_7.

Black, E., S. P. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. P. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski (1991). "A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars". In: *Proceedings of the Workshop on Speech and Natural Language*. HLT '91. Pacific Grove, California: Association for Computational Linguistics, pp. 306–311. DOI: 10.3115/112405.112467.

Blaheta, Don and Eugene Charniak (2000). "Assigning Function Tags to Parsed Text". In: *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. URL: https://www.aclweb.org/anthology/A00-2031.

Bod, Rens (1992). "A Computational Model of Language Performance: Data Oriented Parsing". In: *COLING 1992 Volume 3: The 15th International Conference on Computational Linguistics*. URL: https://www.aclweb.org/anthology/C92-3126.

– (1995). "The Problem of Computing the Most Probable Tree in Data-Oriented Parsing and Stochastic Tree Grammars". In: *Seventh Conference of the European Chapter of the Association for Computational Linguistics*. URL: http://aclweb.org/anthology/E95-1015.

– (2003). "An efficient implementation of a new DOP model". In: *10th Conference of the European Chapter of the Association for Computational Linguistics*. Budapest, Hungary: Association for Computational Linguistics. URL: https://www.aclweb.org/anthology/E03-1005.

Bodirsky, Manuel, Marco Kuhlmann, and Mathias Möhl (2005). "Well-Nested Drawings as Models of Syntactic Structure". In: *Proceedings of the 10th Conference on Formal Grammar (FG) and Ninth Meeting on Mathematics of Language (MOL)*. Edinburgh, UK, pp. 195–203.

Boullier, Pierre (1998). *Proposal for a Natural Language Processing Syntactic Backbone*. Research Report RR-3342. INRIA. URL: https://hal.inria.fr/inria-00073347.

Boyd, Adriane (2007). "Discontinuity Revisited: An Improved Conversion to Context-Free Representations". In: *Proceedings of the Linguistic Annotation Workshop*. Prague, Czech Republic: Association for Computational Linguistics, pp. 41–44. URL: https://www.aclweb.org/anthology/W07-1506.

Brainerd, Walter S. (1968). "The minimalization of tree automata". In: *Information and Control* 13 (5), pp. 484–491. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(68)90917-0.

– (1969). "Tree generating regular systems". In: *Information and Control* 14 (2), pp. 217–231. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(69)90065-5.

Brants, Sabine, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit (2004). "TIGER: Linguistic Interpretation of a German Corpus". In: *Research on Language and Computation* 2 (4), pp. 597–620. ISSN: 1572-8706. DOI: 10.1007/s11168-004-7431-3.

Bresnan, Joan (2001). *Lexical-Functional Syntax*. John Wiley & Sons, Ltd. ISBN: 9781119105664. DOI: 10.1002/9781119105664.

Brown, Peter F., Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer (1992). "Class-Based n-gram Models of Natural Language". In: *Computational Linguistics* 18 (4), pp. 467–480. URL: https://www.aclweb.org/anthology/J92-4003.

Brüggemann, Anne, Makoto Murata, and Derick Wood (2001). *Regular Tree and Regular Hedge Languages over Unranked Alphabets*. Tech. rep. HKUST Theoretical Computer Science Center, pp. 1–29. URL: https://www.cse.ust.hk/tcsc/RR/2001-05.ps.gz.

Buchholz, Sabine and Erwin Marsi (2006). "CoNLL-X Shared Task on Multilingual Dependency Parsing". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City: Association for Computational Linguistics, pp. 149–164. URL: https://www.aclweb.org/anthology/W06-2920.

Büchse, Matthias, Alexander Koller, and Heiko Vogler (2013). "General binarization for parsing and translation". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 145–154. URL: https://www.aclweb.org/anthology/P13-1015.

Burden, Håkan and Peter Ljunglöf (2005). "Parsing Linear Context-Free Rewriting Systems". In: *Proceedings of the Ninth International Workshop on Parsing Technology*. Vancouver, British Columbia: Association for Computational Linguistics, pp. 11–17. URL: https://www.aclweb.org/anthology/W05-1502.

Cahill, Aoife, Mairéad McCarthy, Josef van Genabith, and Andy Way (2002). "Automatic annotation of the Penn-treebank with LFG f-structure information". In: *LREC 2002 Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data*. Las Palmas, Canary Islands.

Cai, Shu, David Chiang, and Yoav Goldberg (2011). "Language-Independent Parsing with Empty Elements". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Portland, Oregon, USA: Association for Computational Linguistics, pp. 212–216. URL: https://www.aclweb.org/anthology/P11-2037.

Carreras, Xavier, Michael Collins, and Terry Koo (2008). "TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-Rich Parsing". In: *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*. Manchester, England: Coling 2008 Organizing Committee, pp. 9–16. URL: https://www.aclweb.org/anthology/W08-2102.

Casacuberta, Francisco and Colin de la Higuera (2000). "Computational Complexity of Problems on Probabilistic Grammars and Transducers". In: *Grammatical Inference: Algorithms and Applications*. Ed. by Arlindo L. Oliveira. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–24. ISBN: 978-3-540-45257-7. DOI: 10.1007/978-3-540-45257-7_2.

Chanod, Jean-Pierre (2001). "Robust Parsing and Beyond". In: *Robustness in Language and Speech Technology*. Ed. by Jean-Claude Junqua and Gertjan van Noord. Dordrecht: Springer Netherlands, pp. 187–204. ISBN: 978-94-015-9719-7. DOI: 10.1007/978-94-015-9719-7_8.

Charniak, Eugene (1996). "Tree-bank Grammars". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. Portland, Oregon: AAAI Press, pp. 1031–1036. ISBN: 0-262-51091-X. URL: https://www.aaai.org/Papers/AAAI/1996/AAAI96-153.pdf.

– (2000). "A Maximum-Entropy-Inspired Parser". In: *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. URL: https://www.aclweb.org/anthology/A00-2018.

Charniak, Eugene and Mark Johnson (2005). "Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 173–180. DOI: 10.3115/1219840.1219862.

Chi, Zhiyi (1999). "Statistical Properties of Probabilistic Context-Free Grammars". In: *Computational Linguistics* 25 (1), pp. 131–160. URL: https://www.aclweb.org/anthology/J99-1004.

Chiang, David (2007). "Hierarchical Phrase-Based Translation". In: *Computational Linguistics* 33 (2), pp. 201–228. ISSN: 0891-2017. DOI: 10.1162/coli.2007.33.2.201.

Chitrao, Mahesh V. and Ralph Grishman (1990). "Statistical Parsing of Messages". In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*. URL: https://www.aclweb.org/anthology/H90-1053.

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language*

Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

Choe, Do Kook and Eugene Charniak (2016). "Parsing as Language Modeling". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2331–2336. DOI: 10.18653/v1/D16-1257.

Chomsky, Noam (1956). "Three models for the description of language". In: *IRE Transactions on Information Theory* 2 (3), pp. 113–124. ISSN: 0096-1000. DOI: 10.1109/TIT.1956.1056813.

– (1959). "On certain formal properties of grammars". In: *Information and Control* 2 (2), pp. 137–167. DOI: 10.1016/S0019-9958(59)90362-6.

Chu, Yoeng-Jin and Tseng-Hong Liu (1965). "On shortest arborescence of a directed graph". In: *Scientia Sinica* 14 (10), pp. 1396–1400.

Clark, Stephen and James R. Curran (2004). "The Importance of Supertagging for Wide-Coverage CCG Parsing". In: *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. Geneva, Switzerland: COLING, pp. 282–288. URL: https://www.aclweb.org/anthology/C04-1041.

– (2007). "Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models". In: *Computational Linguistics* 33 (4), pp. 493–552. DOI: 10.1162/coli.2007.33.4.493.

Clark, Stephen, Julia Hockenmaier, and Mark Steedman (2002). "Building Deep Dependency Structures using a Wide-Coverage CCG Parser". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp. 327–334. DOI: 10.3115/1073083.1073138.

Coavoux, Maximin and Shay B. Cohen (2019). "Discontinuous Constituency Parsing with a Stack-Free Transition System and a Dynamic Oracle". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 204–217. URL: https://www.aclweb.org/anthology/N19-1018.

Coavoux, Maximin and Benoît Crabbé (2017). "Incremental Discontinuous Phrase Structure Parsing with the GAP Transition". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain, pp. 1259–1270. URL: https://www.aclweb.org/anthology/E17-1118.

Coavoux, Maximin, Benoît Crabbé, and Shay B. Cohen (2019). "Unlexicalized Transition-based Discontinuous Constituency Parsing". In: *Transactions of the Association for Computational Linguistics* 7, pp. 73–89. DOI: 10.1162/tacl_a_00255.

Cohen, Shay B., Karl Stratos, Michael Collins, Dean P. Foster, and Lyle Ungar (2012). "Spectral Learning of Latent-Variable PCFGs". In: *Proceedings of the 50th*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Jeju Island, Korea: Association for Computational Linguistics, pp. 223–231. URL: https://www.aclweb.org/anthology/P12-1024.
– (2013). "Experiments with Spectral Learning of Latent-Variable PCFGs". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, pp. 148–157. URL: https://www.aclweb.org/anthology/N13-1015.
– (2014). "Spectral Learning of Latent-Variable PCFGs: Algorithms and Sample Complexity". In: *Journal of Machine Learning Research* 15, pp. 2399–2449. URL: http://jmlr.org/papers/v15/cohen14a.html.
Collins, Michael John (1999). "Head-driven Statistical Models for Natural Language Parsing". http://www.cs.columbia.edu/∼mcollins/papers/thesis.ps. PhD thesis. Philadelphia, PA, USA: University of Pennsylvania. ISBN: 0-599-25874-8.
Corazza, Anna and Giorgio Satta (2006). "Cross-Entropy and Estimation of Probabilistic Context-Free Grammars". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York City, USA: Association for Computational Linguistics, pp. 335–342. URL: http://www.aclweb.org/anthology/N/N06/N06-1043.
Corro, Caio, Joseph Le Roux, and Mathieu Lacroix (2017). "Efficient Discontinuous Phrase-Structure Parsing via the Generalized Maximum Spanning Arborescence". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark, pp. 1644–1654. URL: https://www.aclweb.org/anthology/D17-1172.
Corston-Oliver, Simon and Anthony Aue (2006). "Dependency Parsing with Reference to Slovene, Spanish and Swedish". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City: Association for Computational Linguistics, pp. 196–200. URL: https://www.aclweb.org/anthology/W06-2928.
Courcelle, Bruno and Paul Franchi-Zannettacci (1982). "Attribute grammars and recursive program schemes I". In: *Theoretical Computer Science* 17 (2), pp. 163–191. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(82)90003-2.
Covington, Michael A. (2001). "A Fundamental Algorithm for Dependency Parsing". In: *Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102.
Van Cranenburgh, Andreas (2012). "Efficient parsing with Linear Context-Free Rewriting Systems". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Avignon, France: Association for Computational Linguistics, pp. 460–470. URL: https://www.aclweb.org/anthology/E12-1047.
Van Cranenburgh, Andreas and Rens Bod (2013). "Discontinuous Parsing with an Efficient and Accurate DOP Model". In: *Proceedings of The 13th International Conference on Parsing Technologies (IWPT 2013)*. Nara, Japan: Assocation for

Computational Linguistics, pp. 7–16. URL: https://www.aclweb.org/anthology/W13-5701.

Van Cranenburgh, Andreas, Remko Scha, and Rens Bod (2016). "Data-Oriented Parsing with discontinuous constituents and function tags". In: *Journal of Language Modelling* 4 (1), pp. 57–111. DOI: 10.15398/jlm.v4i1.100.

Van Cranenburgh, Andreas, Remko Scha, and Federico Sangati (2011). "Discontinuous Data-Oriented Parsing: A mildly context-sensitive all-fragments grammar". In: *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*. Dublin, Ireland: Association for Computational Linguistics, pp. 34–44. URL: https://www.aclweb.org/anthology/W11-3805.

Crescenzi, Pierluigi, Daniel Gildea, Andrea Marino, Gianluca Rossi, and Giorgio Satta (2011). "Optimal Head-Driven Parsing Complexity for Linear Context-Free Rewriting Systems". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 450–459. URL: https://www.aclweb.org/anthology/P11-1046.

Cross, James and Liang Huang (2016). "Incremental Parsing with Minimal Features Using Bi-Directional LSTM". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 32–37. DOI: 10.18653/v1/P16-2006.

Culotta, Aron and Jeffrey Sorensen (2004). "Dependency Tree Kernels for Relation Extraction". In: *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. ACL '04. Barcelona, Spain: Association for Computational Linguistics. DOI: 10.3115/1218955.1219009.

Dakota, Daniel (2018). "The Devil is in the Details: Parsing Unknown German Words". In: *Language Technologies for the Challenges of the Digital Age*. Ed. by Georg Rehm and Thierry Declerck. Cham: Springer International Publishing, pp. 23–39. ISBN: 978-3-319-73706-5. DOI: 10.1007/978-3-319-73706-5_3.

De Alencar, Leonel Figueiredo (2017). "A Computational Implementation of Periphrastic Verb Constructions in French". In: *Alfa: Revista de Linguistica* 61, pp. 351–380. ISSN: 1981-5794. DOI: http://dx.doi.org/10.1590/1981-5794-1709-5.

Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39 (1), pp. 1–38. ISSN: 00359246. URL: https://www.jstor.org/stable/2984875.

Deransart, Pierre and Jan Małuszyński (1985). "Relating logic programs and attribute grammars". In: *Journal of Logic Programming* 2 (2), pp. 119–155. DOI: 10.1016/0743-1066(85)90015-9.

– (1989). "A grammatical view of logic programming". In: *Programming Languages Implementation and Logic Programming*. Ed. by P. Deransart, B. Lorho, and

J. Małuszyński. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 219–251. ISBN: 978-3-540-46092-3. DOI: 10.1007/3-540-50820-1_50.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

Dienes, Péter and Amit Dubey (2003). "Antecedent Recovery: Experiments with a Trace Tagger". In: *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pp. 33–40. URL: https://www.aclweb.org/anthology/W03-1005.

Dietze, Toni (2018). "A Formal View on Training of Weighted Tree Automata by Likelihood-Driven State Splitting and Merging". PhD thesis. Dresden, Germany: Technische Universität Dresden. URL: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-341106.

Dietze, Toni and Mark-Jan Nederhof (2015). "Count-based state merging for probabilistic regular tree grammars". In: *Proceedings of the 12th International Conference on Finite State Methods and Natural Language Processing*. Association for Computational Linguistics. URL: https://www.aclweb.org/anthology/W15-4804.

Ding, Yuan and Martha Palmer (2005). "Machine Translation Using Probabilistic Synchronous Dependency Insertion Grammars". In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. ACL '05. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 541–548. DOI: 10.3115/1219840.1219907.

Drewes, Frank, Kilian Gebhardt, and Heiko Vogler (2016). "EM-Training for Weighted Aligned Hypergraph Bimorphisms". In: *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*. Berlin, Germany, pp. 60–69. URL: https://www.aclweb.org/anthology/W16-2407.

Dubey, Amit and Frank Keller (2003). "Probabilistic Parsing for German Using Sister-Head Dependencies". In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan: Association for Computational Linguistics, pp. 96–103. DOI: 10.3115/1075096.1075109.

Duric, Adnan and Fei Song (2011). "Feature Selection for Sentiment Analysis Based on Content and Syntax Models". In: *Proceedings of the 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis (WASSA 2.011)*. Portland, Oregon: Association for Computational Linguistics, pp. 96–103. URL: https://www.aclweb.org/anthology/W11-1712.

Durrett, Greg and Dan Klein (2015). "Neural CRF Parsing". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th*

International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Beijing, China: Association for Computational Linguistics, pp. 302–312. DOI: 10.3115/v1/P15-1030.

Edmonds, Jack (1967). "Optimum branchings". In: *Journal of Research of the National Bureau of Standards* 71B (4), pp. 233–240. DOI: 10.6028/jres.071b.032.

Eisner, Jason M. (1996). "Three New Probabilistic Models for Dependency Parsing: An Exploration". In: *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. URL: https://www.aclweb.org/anthology/C96-1058.

Evang, Kilian and Laura Kallmeyer (2011). "PLCFRS Parsing of English Discontinuous Constituents". In: *Proceedings of the 12th International Conference on Parsing Technologies*. Dublin, Ireland, pp. 104–116. ISBN: 978-1-932432-04-6. URL: https://www.aclweb.org/anthology/W11-2913.

Fernández-González, Daniel and André F. T. Martins (2015). "Parsing as Reduction". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China, pp. 1523–1533. URL: https://www.aclweb.org/anthology/P15-1147.

Ferraro, Francis, Benjamin Van Durme, and Matt Post (2012). "Toward Tree Substitution Grammars with Latent Annotations". In: *Proceedings of the NAACL-HLT Workshop on the Induction of Linguistic Structure*. Montréal, Canada: Association for Computational Linguistics, pp. 23–30. URL: https://www.aclweb.org/anthology/W12-1904.

Fischer, Michael J. (1968). "Grammars with Macro-like Productions". In: *Proceedings of the 9th Annual Symposium on Switching and Automata Theory (Swat 1968)*. SWAT '68. Washington, DC, USA: IEEE Computer Society, pp. 131–142. DOI: 10.1109/SWAT.1968.12.

Flickinger, Dan (2000). "On Building a More Efficient Grammar by Exploiting Types". In: *Nat. Lang. Eng.* 6 (1), pp. 15–28. ISSN: 1351-3249. DOI: 10.1017/S1351324900002370.

Fried, Daniel and Dan Klein (2018). "Policy Gradient as a Proxy for Dynamic Oracles in Constituency Parsing". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 469–476. URL: https://www.aclweb.org/anthology/P18-2075.

Gabbard, Ryan, Seth Kulick, and Mitchell Marcus (2006). "Fully Parsing the Penn Treebank". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York City, USA: Association for Computational Linguistics, pp. 184–191. URL: https://www.aclweb.org/anthology/N06-1024.

Gebhardt, Kilian (2018). "Generic refinement of expressive grammar formalisms with an application to discontinuous constituent parsing". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico,

USA: Association for Computational Linguistics, pp. 3049–3063. URL: http://www.aclweb.org/anthology/C18-1258.
– (2020). "Advances in using Grammars with Latent Annotations for Discontinuous Parsing". In: *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task*. Virtual Meeting. Association for Computational Linguistics, pp. 91–97. URL: https://www.aclweb.org/anthology/2020.iwpt-1.9.pdf.
Gebhardt, Kilian, Mark-Jan Nederhof, and Heiko Vogler (2017). "Hybrid Grammars for Parsing of Discontinuous Phrase Structures and Non-Projective Dependency Structures". In: *Computational Linguistics* 43 (3), pp. 465–520. DOI: 10.1162/COLI_a_00291.
Gebhardt, Kilian and Johannes Osterholzer (2015). "A Direct Link between Tree-Adjoining and Context-Free Tree Grammars". In: *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing 2015 (FSMNLP 2015 Düsseldorf)*. Association for Computational Linguistics. URL: https://www.aclweb.org/anthology/W15-4805.
Gécseg, Ferenc and Magnus Steinby (1984). *Tree Automata*. Budapest: Akadémiai Kiadó. arXiv: 1509.06233. URL: http://arxiv.org/abs/1509.06233.
Gildea, Daniel (2010). "Optimal Parsing Strategies for Linear Context-Free Rewriting Systems". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California: Association for Computational Linguistics, pp. 769–776. URL: https://www.aclweb.org/anthology/N10-1118.
Goguen, Joseph A., James W. Thatcher, Eric G. Wagner, and Jesse B. Wright (1977). "Initial algebra semantics and continuous algebras". In: *J. ACM* 24 (1), pp. 68–95. ISSN: 0004-5411. DOI: 10.1145/321992.321997.
Goldberg, Yoav (2017). "Neural Network Methods for Natural Language Processing". In: *Synthesis Lectures on Human Language Technologies* 10 (1), pp. 1–309. DOI: 10.2200/S00762ED1V01Y201703HLT037.
– (2019). *Assessing BERT's Syntactic Abilities*. arXiv: 1901.05287 [cs.CL].
Goldberg, Yoav and Michael Elhadad (2010). "An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California: Association for Computational Linguistics, pp. 742–750. URL: https://www.aclweb.org/anthology/N10-1115.
Goldberg, Yoav and Joakim Nivre (2013). "Training Deterministic Parsers with Non-Deterministic Oracles". In: *Transactions of the Association for Computational Linguistics* 1, pp. 403–414. DOI: 10.1162/tacl_a_00237.
Gómez-Rodríguez, Carlos, Marco Kuhlmann, Giorgio Satta, and David Weir (2009). "Optimal Reduction of Rule Length in Linear Context-Free Rewriting Systems". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Boulder, Colorado: Association for Computational Linguistics, pp. 539–547. URL: https://www.aclweb.org/anthology/N09-1061.

Gómez-Rodríguez, Carlos, Francesco Sartorio, and Giorgio Satta (2014). "A Polynomial-Time Dynamic Oracle for Non-Projective Dependency Parsing". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 917–927. DOI: 10.3115/v1/D14-1099.

Gómez-Rodríguez, Carlos and Giorgio Satta (2009). "An Optimal-Time Binarization Algorithm for Linear Context-Free Rewriting Systems with Fan-Out Two". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, pp. 985–993. URL: https://www.aclweb.org/anthology/P09-1111.

Gómez-Rodríguez, Carlos and David Vilares (2018). "Constituent Parsing as Sequence Labeling". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 1314–1324. URL: https://www.aclweb.org/anthology/D18-1162.

Goodman, Joshua (1996a). "Parsing Algorithms and Metrics". In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz, California, USA: Association for Computational Linguistics, pp. 177–183. DOI: 10.3115/981863.981887.

– (1996b). "Efficient Algorithms for Parsing the DOP Model". In: *Conference on Empirical Methods in Natural Language Processing*. URL: http://aclweb.org/anthology/W96-0214.

– (2003). "Efficient Parsing of DOP with PCFG-reductions". In: *Data-Oriented Parsing*. Ed. by Rens Bod, Khalil Sima'an, and Remko Scha. Stanford, CA, USA: CSLI Publications, pp. 125–146. ISBN: 1575864355.

Gorman, Kyle and Steven Bedrick (2019). "We Need to Talk about Standard Splits". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 2786–2791. DOI: 10.18653/v1/P19-1267.

Gorn, Saul (1967). "Explicit definitions and linguistic dominoes". In: *Systems and Computer Science*. Ed. by J. Hart and S. Takasu. Toronto: University of Toronto Press, pp. 77–115.

Groschwitz, Jonas, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller (2018). "AMR dependency parsing with a typed semantic algebra". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1831–1841. DOI: 10.18653/v1/P18-1170.

Grune, Dick and Ceriel J. H. Jacobs (2008). "Parsing as Intersection". In: *Parsing Techniques: A Practical Guide*. New York, NY: Springer New York, pp. 425–442. ISBN: 978-0-387-68954-8. DOI: 10.1007/978-0-387-68954-8_13.

Grünewald, Stefan, Sophie Henning, and Alexander Koller (2018). "Generalized chart constraints for efficient PCFG and TAG parsing". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 626–631. DOI: 10.18653/v1/P18-2099.

Habel, Annegret (1992). *Hyperedge Replacement: Grammars and Languages*. Vol. 643. Lecture Notes in Computer Science. Springer. DOI: 10.1007/BFb0013875.

Hajič, Jan, Alena Böhmová, Eva Hajičová, and Barbora Vidová-Hladká (2000). "The Prague Dependency Treebank: A Three-Level Annotation Scenario". In: *Treebanks: Building and Using Parsed Corpora*. Ed. by A. Abeillé. Amsterdam: Kluwer, pp. 103–127.

Hall, Johan and Joakim Nivre (2008). "Parsing Discontinuous Phrase Structure with Grammatical Functions". In: *Advances in Natural Language Processing*. Ed. by Bengt Nordström and Aarne Ranta. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 169–180. ISBN: 978-3-540-85287-2. DOI: 10.1007/978-3-540-85287-2_17.

Hall, Johan, Joakim Nivre, and Jens Nilsson (2006). "Discriminative Classifiers for Deterministic Dependency Parsing". In: *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*. Sydney, Australia: Association for Computational Linguistics, pp. 316–323. URL: https://www.aclweb.org/anthology/P06-2041.

De la Higuera, Colin (1997). "Characteristic Sets for Polynomial Grammatical Inference". In: *Machine Learning* 27 (2), pp. 125–138. ISSN: 1573-0565. DOI: 10.1023/A:1007353007695.

De la Higuera, Colin and Jose Oncina (2013a). "The most probable string: an algorithmic study". In: *Journal of Logic and Computation* 24 (2), pp. 311–330. ISSN: 0955-792X. DOI: 10.1093/logcom/exs049.

– (2013b). "Computing the Most Probable String with a Probabilistic Finite State Machine". In: *Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing*. St Andrews, Scotland: Association for Computational Linguistics, pp. 1–8. URL: http://www.aclweb.org/anthology/W13-1801.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Comput.* 9 (8), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Hockenmaier, Julia and Mark Steedman (2002). "Acquiring Compact Lexicalized Grammars from a Cleaner Treebank". In: *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*. Las Palmas, Canary Islands - Spain: European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2002/pdf/263.pdf.

Holan, Tomas, Vladislav Kubon, Karel Oliva, and Martin Platek (1998). "Two Useful Measures of Word Order Complexity". In: *Processing of Dependency-Based Grammars*, pp. 21–28. URL: https://www.aclweb.org/anthology/W98-0503.

Hsu, Yu-Yin (2010). "Comparing conversions of discontinuity in PCFG parsing". In: *Ninth International Workshop on Treebanks and Linguistic Theories*. Ed. by Markus Dickinson, Kaili Müürisep, and Marco Passarotti. Tartu, Estonia: Northern European Association for Language Technology, pp. 103–113. URL: http://hdl.handle.net/10062/15954.

Huang, Liang (2008a). "Advanced Dynamic Programming in Semiring and Hypergraph Frameworks". In: *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*. Manchester, UK: Coling 2008 Organizing Committee, pp. 1–18. URL: https://www.aclweb.org/anthology/C08-5001.

– (2008b). "Forest Reranking: Discriminative Parsing with Non-Local Features". In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 586–594. URL: https://www.aclweb.org/anthology/P08-1067.

Huang, Liang and David Chiang (2005). "Better K-best Parsing". In: *Proceedings of the Ninth International Workshop on Parsing Technology*. Vancouver, British Columbia, Canada, pp. 53–64. URL: https://www.aclweb.org/anthology/W05-1506.

Huang, Zhongqiang and Mary Harper (2009). "Self-Training PCFG Grammars with Latent Annotations Across Languages". In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, pp. 832–841. URL: https://www.aclweb.org/anthology/D09-1087.

Huang, Zhongqiang, Mary Harper, and Slav Petrov (2010). "Self-Training with Products of Latent Variable Grammars". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, pp. 12–22. URL: https://www.aclweb.org/anthology/D10-1002.

Jazayeri, Mehdi, William F. Ogden, and William C. Rounds (1975). "The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars". In: *Commun. ACM* 18 (12), pp. 697–706. ISSN: 0001-0782. DOI: 10.1145/361227.361231.

Johnson, Mark (1998). "PCFG Models of Linguistic Tree Representations". In: *Computational Linguistics* 24 (4), pp. 613–632. URL: https://www.aclweb.org/anthology/J98-4004.

– (2002). "A Simple Pattern-matching Algorithm for Recovering Empty Nodes and their Antecedents". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp. 136–143. DOI: 10.3115/1073083.1073107.

Johnson, Mark, Thomas Griffiths, and Sharon Goldwater (2007). "Bayesian Inference for PCFGs via Markov Chain Monte Carlo". In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. Rochester, New York: Association for Computational Linguistics, pp. 139–146. URL: https://www.aclweb.org/anthology/N07-1018.

Joshi, Aravind K. (1985). "Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?" In: *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*. Ed. by David R. Dowty, Lauri Karttunen, and Arnold M.Editors Zwicky. Studies in Natural Language Processing. Cambridge University Press, pp. 206–250. DOI: 10.1017/CBO9780511597855.007.

Joshi, Aravind K., Leon S. Levy, and Masako Takahashi (1975). "Tree Adjunct Grammars". In: *J. Comput. Syst. Sci.* 10 (1), pp. 136–163. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(75)80019-5.

Joshi, Aravind K. and Yves Schabes (1992). "Tree-adjoining grammars and lexicalized grammars". In: *Tree Automata and Languages*. Ed. by Maurice Nivat and Andreas Podelski. North Holland: Elsevier Science. ISBN: 9780080934235.

Kahane, Sylvain, Alexis Nasr, and Owen Rambow (1998). "Pseudo-Projectivity: A Polynomially Parsable Non-Projective Dependency Grammar". In: *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, pp. 646–652. URL: https://www.aclweb.org/anthology/C98-1102.

Kallmeyer, Laura (2010). *Parsing Beyond Context-Free Grammars*. 1st. Springer Publishing Company, Incorporated. ISBN: 9783642148453.

Kallmeyer, Laura and Marco Kuhlmann (2012). "A Formal Model for Plausible Dependencies in Lexicalized Tree Adjoining Grammar". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*. Paris, France, pp. 108–116. URL: https://www.aclweb.org/anthology/W12-4613.

Kallmeyer, Laura and Wolfgang Maier (2010). "Data-Driven Parsing with Probabilistic Linear Context-Free Rewriting Systems". In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China: Coling 2010 Organizing Committee, pp. 537–545. URL: https://www.aclweb.org/anthology/C10-1061.

– (2013). "Data-driven Parsing using Probabilistic Linear Context-Free Rewriting Systems". In: *Computational Linguistics* 39 (1), pp. 87–119. DOI: 10.1162/COLI_a_00136.

Kaplan, Ronald and Joan Bresnan (1982). "Lexical-Functional Grammar: A Formal System for Grammatical Representation". In: *The Mental Representation of Grammatical Relations*. Ed. by Joan Bresnan. MIT Press. Chap. 4, pp. 173–281.

Kay, Martin (1996). "Chart Generation". In: *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*. ACL '96. Santa Cruz, California:

Association for Computational Linguistics, pp. 200–204. DOI: 10.3115/981863.981890.

Kennedy, Ken and Scott K. Warren (1976). "Automatic Generation of Efficient Evaluators for Attribute Grammars". In: *Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages*. POPL '76. Atlanta, Georgia: ACM, pp. 32–49. DOI: 10.1145/800168.811538.

Kepser, Stephan and Jim Rogers (2011). "The Equivalence of Tree Adjoining Grammars and Monadic Linear Context-free Tree Grammars". In: *Journal of Logic, Language and Information* 20 (3), pp. 361–384. ISSN: 1572-9583. DOI: 10.1007/s10849-011-9134-0.

Kiperwasser, Eliyahu and Yoav Goldberg (2016). "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations". In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327. DOI: 10.1162/tacl_a_00101.

Kitaev, Nikita and Dan Klein (2018). "Constituency Parsing with a Self-Attentive Encoder". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 2676–2686. DOI: 10.18653/v1/P18-1249.

– (2019). "Tetra-Tagging: Word-Synchronous Parsing with Linear-Time Inference". In: *CoRR* abs/1904.09745. arXiv: 1904.09745. URL: http://arxiv.org/abs/1904.09745.

Klein, Dan and Christopher D. Manning (2003). "Accurate Unlexicalized Parsing". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. Sapporo, Japan, pp. 423–430. URL: https://www.aclweb.org/anthology/P03-1054.

Knaster, Bronisław (1928). "Un théorème sur les fonctions d'ensembles". In: *Annales de la Société Polonaise de Mathématique* 6, pp. 133–134.

Knuth, Donald E. (1968). "Semantics of context-free languages". English. In: *Mathematical Systems Theory* 2 (2), pp. 127–145. ISSN: 0025-5661. DOI: 10.1007/BF01692511.

– (1977). "A generalization of Dijkstra's algorithm". In: *Information Processing Letters* 6 (1), pp. 1–5. ISSN: 0020-0190. DOI: 10.1016/0020-0190(77)90002-3.

Koller, Alexander and Marco Kuhlmann (2011). "A Generalized View on Parsing and Translation". In: *Proceedings of the 12th International Conference on Parsing Technologies*. Dublin, Ireland: Association for Computational Linguistics, pp. 2–13. URL: http://www.aclweb.org/anthology/W11-2902.

Koo, Terry, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag (2010). "Dual Decomposition for Parsing with Non-Projective Head Automata". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, pp. 1288–1298. URL: https://www.aclweb.org/anthology/D10-1125.

Kuhlmann, Marco (2013). "Mildly Non-Projective Dependency Grammar". In: *Computational Linguistics* 39 (2), pp. 355–387. DOI: 10.1162/COLI_a_00125.

Kuhlmann, Marco, Carlos Gómez-Rodríguez, and Giorgio Satta (2011). "Dynamic Programming Algorithms for Transition-Based Dependency Parsers". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 673–682. URL: https://www.aclweb.org/anthology/P11-1068.

Kuhlmann, Marco, Alexander Koller, and Giorgio Satta (2015). "Lexicalization and Generative Power in CCG". In: *Computational Linguistics* 41 (2), pp. 187–219. DOI: 10.1162/COLI_a_00219.

Kuhlmann, Marco and Joachim Niehren (2008). "Logics and Automata for Totally Ordered Trees". In: *Rewriting Techniques and Applications*. Ed. by Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 217–231. ISBN: 978-3-540-70590-1.

Kuhlmann, Marco and Joakim Nivre (2010). "Transition-Based Techniques for Non-Projective Dependency Parsing". In: *Northern European Journal of Language Technology* 2, pp. 1–19. DOI: 10.3384/nejlt.2000-1533.10211.

Kuhlmann, Marco and Giorgio Satta (2009). "Treebank Grammar Techniques for Non-Projective Dependency Parsing". In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. Athens, Greece: Association for Computational Linguistics, pp. 478–486. URL: https://www.aclweb.org/anthology/E09-1055.

Kulmizev, Artur, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre (2019). "Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing - A Tale of Two Parsers Revisited". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 2755–2768. DOI: 10.18653/v1/D19-1277.

Kunze, Jürgen (1968). "The treatment of non-projective structures in the syntactic analysis and synthesis of english and german". In: *Computational Linguistics* 7, pp. 67–77.

Lafferty, John D., Andrew McCallum, and Fernando Pereira (2001). "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 282–289. ISBN: 1-55860-778-1. URL: https://repository.upenn.edu/cis_papers/159/.

Lang, Bernard (1994). "Regcognition can be harden than parsing". In: *Computational Intelligence* 10 (4), pp. 486–494. DOI: 10.1111/j.1467-8640.1994.tb00011.x.

Lari, Karim and Steve J. Young (1990). "The estimation of stochastic context-free grammars using the Inside-Outside algorithm". In: *Computer Speech & Language* 4 (1), pp. 35–56. ISSN: 0885-2308. DOI: 10.1016/0885-2308(90)90022-X.

Lautemann, Clemens (1990). "The complexity of graph languages generated by hyperedge replacement". In: *Acta Informatica* 27 (5), pp. 399–421. ISSN: 1432-0525. DOI: 10.1007/BF00289017.

Lê, Minh and Antske Fokkens (2017). "Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 677–687. URL: https://www.aclweb.org/anthology/E17-1064.

Levy, Roger and Christopher D. Manning (2004). "Deep Dependencies from Context-Free Statistical Parsers: Correcting the Surface Dependency Approximation". In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*. Barcelona, Spain, pp. 327–334. DOI: 10.3115/1218955.1218997.

Lewis, Mike and Mark Steedman (2014). "Improved CCG Parsing with Semi-supervised Supertagging". In: *Transactions of the Association for Computational Linguistics* 2, pp. 327–338. DOI: 10.1162/tacl_a_00186.

De Lhoneux, Miryam, Sara Stymne, and Joakim Nivre (2017). "Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle". In: *Proceedings of the 15th International Conference on Parsing Technologies*. Pisa, Italy: Association for Computational Linguistics, pp. 99–104. URL: https://www.aclweb.org/anthology/W17-6314.

Li, Zuchao, Jiaxun Cai, Shexia He, and Hai Zhao (2018). "Seq2seq Dependency Parsing". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 3203–3214. URL: https://www.aclweb.org/anthology/C18-1271.

Liang, Percy, Michael I. Jordan, and Dan Klein (2010). *Probabilistic grammars and hierarchical Dirichlet processes*. Ed. by Anthony O'Hagan and Mike West. DOI: 10.1093/oxfordhb/9780198703174.013.27.

Liang, Percy, Slav Petrov, Michael Jordan, and Dan Klein (2007). "The Infinite PCFG Using Hierarchical Dirichlet Processes". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 688–697. URL: https://www.aclweb.org/anthology/D07-1072.

Libkin, Leonid (2006). "Logics for Unranked Trees: An Overview". In: *Logical Methods in Computer Science* 2 (3). DOI: 10.2168/LMCS-2(3:2)2006.

Lichte, Timm (2007). "An MCTAG with Tuples for Coherent Constructions in German". In: *Proceedings of the 12th Conference on Formal Grammar*. 4-5 August 2007. Dublin, Ireland.

Lin, Chu-Cheng, Hao Zhu, Matthew R. Gormley, and Jason Eisner (2019). "Neural Finite-State Transducers: Beyond Rational Relations". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 272–283. URL: https://www.aclweb.org/anthology/N19-1024.

Linzen, Tal, Emmanuel Dupoux, and Yoav Goldberg (2016). "Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies". In: *Transactions of the Association for Computational Linguistics* 4, pp. 521–535. DOI: 10.1162/tacl_a_00115.

Lopez, Adam (2008). "Statistical Machine Translation". In: *ACM Comput. Surv.* 40 (3), 8:1–8:49. ISSN: 0360-0300. DOI: 10.1145/1380584.1380586.

Maier, Wolfgang (2015). "Discontinuous Incremental Shift-reduce Parsing". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China, pp. 1202–1212. URL: https://www.aclweb.org/anthology/P15-1116.

Maier, Wolfgang, Miriam Kaeshammer, and Laura Kallmeyer (2012). "PLCFRS Parsing Revisited: Restricting the Fan-Out to Two". In: *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*. Paris, France, pp. 126–134. URL: https://www.aclweb.org/anthology/W12-4615.

Maier, Wolfgang and Laura Kallmeyer (2010). "Discontinuity and Non-Projectivity: Using Mildly Context-Sensitive Formalisms for Data-Driven Parsing". In: *Proceedings of the 10th International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+10)*. Yale University: Linguistic Department, Yale University, pp. 119–126. URL: https://www.aclweb.org/anthology/W10-4415.

Maier, Wolfgang and Timm Lichte (2016). "Discontinuous parsing with continuous trees". In: *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*. San Diego, California, pp. 47–57. URL: https://www.aclweb.org/anthology/W16-0906.

Maier, Wolfgang and Anders Søgaard (2008). "Treebanks and Mild Context-Sensitivity". In: *Proceedings of the 13th Conference on Formal Grammar (FG-2008)*. Hamburg, Germany: CSLI Publications, pp. 61–76. URL: https://cslipublications.stanford.edu/FG/2008/maier.pdf.

Maletti, Andreas and Giorgio Satta (2009). "Parsing Algorithms based on Tree Automata". In: *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*. Paris, France: Association for Computational Linguistics, pp. 1–12. URL: http://www.aclweb.org/anthology/W09-3801.

Manning, Christopher D. (2011). "Part-of-speech Tagging from 97% to 100%: Is It Time for Some Linguistics?" In: *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing*. CICLing'11. Tokyo, Japan: Springer-Verlag, pp. 171–189. ISBN: 978-3-642-19399-6. URL: http://dl.acm.org/citation.cfm?id=1964799.1964816.

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky (2014). "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, pp. 55–60. DOI: 10.3115/v1/P14-5010.

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19 (2), pp. 313–330. URL: http://aclweb.org/anthology/J93-2004.

Marinov, Svetoslav (2007). "Covington Variations". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 1144–1148. URL: https://www.aclweb.org/anthology/D07-1125.

Martins, Bruno and Mário J. Silva (2004). "Spelling Correction for Search Engine Queries". In: *Advances in Natural Language Processing*. Ed. by José Luis Vicedo, Patricio Martínez-Barco, Rafael Muńoz, and Maximiliano Saiz Noeda. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 372–383. ISBN: 978-3-540-30228-5.

Marvin, Rebecca and Tal Linzen (2018). "Targeted Syntactic Evaluation of Language Models". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 1192–1202. DOI: 10.18653/v1/D18-1151.

Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii (2005). "Probabilistic CFG with Latent Annotations". In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Ann Arbor, Michigan, pp. 75–82. DOI: 10.3115/1219840.1219850.

May, Jonathan and Kevin Knight (2006). "A Better N-Best List: Practical Determinization of Weighted Finite Tree Automata". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. URL: http://aclweb.org/anthology/N06-1045.

McCawley, James D. (1982). "Parentheticals and Discontinuous Constituent Structure". In: *Linguistic Inquiry* 13 (1), pp. 91–106. ISSN: 00243892, 15309150. URL: http://www.jstor.org/stable/4178261.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic (2005). "Non-Projective Dependency Parsing using Spanning Tree Algorithms". In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Vancouver, British Columbia,

Canada: Association for Computational Linguistics, pp. 523–530. URL: https://www.aclweb.org/anthology/H05-1066.

Meinert, Pius Friedrich (2019). "The Problem of Computing the Most Probable Tree of a Probabilistic Tree Automaton". MA thesis. Dresden, Germany: Technische Universität Dresden. URL: http://www.orchid.inf.tu-dresden.de/gdp/diploma_theses/Masterarbeit_Meinert.pdf.

Mel'čuk, Igor Aleksandrovic (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Mohri, Mehryar, Fernando Pereira, and Michael Riley (2000). "The design principles of a weighted finite-state transducer library". In: *Theoretical Computer Science* 231 (1), pp. 17–32. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(99)00014-6.

Müller, Stefan (2016). *Grammatical Theory: From Transformational Grammar to Constraint-Based Approaches*. Textbooks in Language Sciences 1. Berlin: Language Science Press. DOI: 10.17169/langsci.b25.167.

Müller, Stefan and Walter Kasper (2000). "HPSG Analysis of German". In: *Verbmobil: Foundations of Speech-to-Speech Translation*. Ed. by Wolfgang Wahlster. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 238–253. ISBN: 978-3-662-04230-4. DOI: 10.1007/978-3-662-04230-4_17.

Nakanishi, Hiroko, Yusuke Miyao, and Jun'ichi Tsujii (2004). "Using Inverse Lexical Rules to Acquire a Wide-coverage Lexicalized Grammar". In: *IJCNLP 2004 Workshop Beyond Shallow Analysis – Formalisms and Statistical Modeling for Deep Analysis*. Sanya City, Hainan Island, China.

Nederhof, Mark-Jan (2003). "Weighted Deductive Parsing and Knuth's Algorithm". In: *Computational Linguistics* 29 (1), pp. 135–143. DOI: 10.1162/089120103321337467.

Nederhof, Mark-Jan and Giorgio Satta (2003). "Probabilistic Parsing as Intersection". In: *Proceedings of the Eighth International Conference on Parsing Technologies*. Nancy, France, pp. 137–148. URL: https://www.aclweb.org/anthology/W03-3016.

– (2004). "Kullback-Leibler Distance Between Probabilistic Context-free Grammars and Probabilistic Finite Automata". In: *Proceedings of the 20th International Conference on Computational Linguistics*. Geneva, Switzerland. DOI: 10.3115/1220355.1220366.

– (2008). "Computing Partition Functions of PCFGs". In: *Research on Language and Computation* 6 (2), pp. 139–162. ISSN: 1572-8706. DOI: 10.1007/s11168-008-9052-8.

Nederhof, Mark-Jan and Heiko Vogler (2014). "Hybrid Grammars for Discontinuous Parsing". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland, pp. 1370–1381. URL: https://www.aclweb.org/anthology/C14-1130.

Nivre, Joakim (2003). "An Efficient Algorithm for Projective Dependency Parsing". In: *Proceedings of the Eighth International Conference on Parsing Technologies*. Nancy, France, pp. 149–160. URL: https://www.aclweb.org/anthology/W03-3017.

– (2008). "Algorithms for Deterministic Incremental Dependency Parsing". In: *Computational Linguistics* 34 (4), pp. 513–553. DOI: 10.1162/coli.07-056-R1-07-027.

– (2009). "Non-Projective Dependency Parsing in Expected Linear Time". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, pp. 351–359. URL: https://www.aclweb.org/anthology/P09-1040.

– (2019). *Is the End of Supervised Parsing in Sight? Twelve Years Later*. Invited talk at EurNLP. London, UK. URL: https://cl.lingfil.uu.se/~nivre/docs/NivreEurNLP2019.pdf.

Nivre, Joakim and Jens Nilsson (2005). "Pseudo-Projective Dependency Parsing". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 99–106. DOI: 10.3115/1219840.1219853.

van Noord, Gertjan (2009). "Huge Parsed Corpora in LASSY". In: *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories (TLT 7)*. Groningen, The Netherlands.

Noreen, Eric W. (1989). *Computer-Intensive Methods for Testing Hypotheses: An Introduction*. Wiley New York. ISBN: 978-0-471-61136-3.

Padó, Sebastian (2006). *User's guide to sigf: Significance testing by approximate randomisation*. URL: https://nlpado.de/~sebastian/software/sigf.shtml.

Pereira, Fernando and David H. D. Warren (1983). "Parsing as Deduction". In: *21st Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, pp. 137–144. DOI: 10.3115/981311.981338.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

Petrov, Slav (2010). "Products of Random Latent Variable Grammars". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California: Association for Computational Linguistics, pp. 19–27. URL: https://www.aclweb.org/anthology/N10-1003.

Petrov, Slav, Leon Barrett, Romain Thibaux, and Dan Klein (2006). "Learning Accurate, Compact, and Interpretable Tree Annotation". In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*. Sydney, Australia, pp. 433–440. DOI: 10.3115/1220175.1220230.

Petrov, Slav and Dan Klein (2007). "Improved Inference for Unlexicalized Parsing". In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. Rochester, New York, pp. 404–411. URL: https://www.aclweb.org/anthology/N07-1051.

– (2008). "Sparse Multi-Scale Grammars for Discriminative Latent Variable Parsing". In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, pp. 867–876. URL: https://www.aclweb.org/anthology/D08-1091.

Pollard, Carl and Ivan A. Sag (1994). *Head-driven Phrase Structure Grammar*. Chicago: University of Chicago Press. ISBN: 0-226-67446-0.

Prescher, Detlef (2004). *A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilistic Context-Free Grammars*. arXiv: cs/0412015 [cs.CL].

Prescher, Detlef, Remko Scha, Khalil Sima'an, and Andreas Zollmann (2003). "On the Statistical Consistency of DOP Estimators". In: *Computational Linguistics in the Netherlands 2003, December 19, Centre for Dutch Language and Speech, University of Antwerp*. URL: http://www.cnts.ua.ac.be/clin2003/proc/07Prescher.pdf.

Räihä, Kari -Jouko and Mikko Saarinen (1982). "Testing Attribute Grammars for Circularity". In: *Acta Inf.* 17 (2), pp. 185–192. ISSN: 0001-5903. DOI: 10.1007/BF00288969.

Rambow, Owen (1994). "Formal and computational aspects of natural language syntax". PhD thesis. University of Pennsylvania. URL: http://repository.upenn.edu/ircs_reports/154.

Ranta, Aarne (2011). *Grammatical Framework: Programming with Multilingual Grammars*. Center for the Study of Language and Information/SRI.

Rehbein, Ines and Josef van Genabith (2009). "Automatic acquisition of LFG resources for German - as good as it gets". In: *Lexical Functional Grammar 2009*. Cambridge, UK.

Ruprecht, Thomas and Tobias Denkinger (2019). "Implementation of a Chomsky-Schützenberger n-best parser for weighted multiple context-free grammars". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 178–191. DOI: 10.18653/v1/N19-1016.

Sag, Ivan A. (2010). "ENGLISH FILLER-GAP CONSTRUCTIONS". In: *Language* 86 (3), pp. 486–545. ISSN: 00978507, 15350665. URL: http://www.jstor.org/stable/40961690.

Sagot, Benoît and Giorgio Satta (2010). "Optimal Rank Reduction for Linear Context-Free Rewriting Systems with Fan-Out Two". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 525–533. URL: https://www.aclweb.org/anthology/P10-1054.

Saluja, Avneesh, Chris Dyer, and Shay B. Cohen (2014). "Latent-Variable Synchronous CFGs for Hierarchical Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1953–1964. DOI: 10.3115/v1/D14-1210.

Sangati, Federico and Willem Zuidema (2011). "Accurate Parsing with Compact Tree-Substitution Grammars: Double-DOP". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 84–95. URL: http://www.aclweb.org/anthology/D11-1008.

Satta, Giorgio and Enoch Peserico (2005). "Some Computational Complexity Results for Synchronous Context-Free Grammars". In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 803–810. URL: https://www.aclweb.org/anthology/H05-1101.

Schiller, Anne, Simone Teufel, Christine Stöckert, and Christine Thielen (1999). *Guidelines für das Tagging deutscher Textcorpora mit STTS. (Kleines und großes Tagset)*. Tech. rep. Universität Stuttgart, Institut für maschinelle Sprachverarbeitung; Universität Tübingen, Seminar für Sprachwissenschaft. URL: http://www.sfs.uni-tuebingen.de/resources/stts-1999.pdf.

Schmid, Helmut (2006). "Trace Prediction and Recovery with Unlexicalized PCFGs and Slash Features". In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Sydney, Australia: Association for Computational Linguistics, pp. 177–184. DOI: 10.3115/1220175.1220198.

Seddah, Djamé, Sandra Kübler, and Reut Tsarfaty (2014). "Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages". In: *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*. Dublin, Ireland, pp. 103–109. URL: https://www.aclweb.org/anthology/W14-6111.

Seki, Hiroyuki and Yuki Kato (2008). "On the Generative Power of Multiple Context-Free Grammars and Macro Grammars". In: *IEICE - Transactions on Information and Systems* E91-D (2), pp. 209–221. ISSN: 0916-8532. DOI: 10.1093/ietisy/e91-d.2.209.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami (1991). "On multiple context-free grammars". In: *Theoretical Computer Science* 88 (2), pp. 191–229. ISSN: 0304-3975. DOI: 10.1016/0304-3975(91)90374-B.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). "Edinburgh Neural Machine Translation Systems for WMT 16". In: *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. Berlin, Germany: Association for Computational Linguistics, pp. 371–376. DOI: 10.18653/v1/W16-2323.

Shen, Yikang, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio (2018). "Straight to the Tree: Constituency Parsing with Neural Syntactic Distance". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1171–1180. DOI: 10.18653/v1/P18-1108.

Shieber, Stuart M. (1985). "Evidence against the context-freeness of natural language". In: *Linguistics and Philosophy* 8 (3), pp. 333–343. ISSN: 1573-0549. DOI: 10.1007/BF00630917.

Shieber, Stuart M. and Yves Schabes (1990). "Synchronous Tree-adjoining Grammars". In: *Proceedings of the 13th Conference on Computational Linguistics - Volume 3*. COLING '90. Helsinki, Finland: Association for Computational Linguistics, pp. 253–258. ISBN: 952-90-2028-7. DOI: 10.3115/991146.991191.

Shindo, Hiroyuki, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata (2012). "Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Jeju Island, Korea: Association for Computational Linguistics, pp. 440–448. URL: https://www.aclweb.org/anthology/P12-1046.

Sima'an, Khalil (2002). "Computational Complexity of Probabilistic Disambiguation". In: *Grammars* 5 (2), pp. 125–151. ISSN: 1572-848X. DOI: 10.1023/A:1016340700671.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit (1997). "An Annotation Scheme for Free Word Order Languages". In: *Fifth Conference on Applied Natural Language Processing*. Washington, DC, USA: Association for Computational Linguistics, pp. 88–95. DOI: 10.3115/974557.974571.

Snow, Rion, Daniel Jurafsky, and Andrew Y. Ng (2005). "Learning Syntactic Patterns for Automatic Hypernym Discovery". In: *Advances in Neural Information Processing Systems*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou, pp. 1297–1304. URL: http://ilpubs.stanford.edu:8090/665/.

Stanojević, Miloš and Raquel Garrido Alhama (2017). "Neural Discontinuous Constituency Parsing". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark, pp. 1666–1676. URL: https://www.aclweb.org/anthology/D17-1174.

Steedman, Mark (1987). "Combinatory grammars and parasitic gaps". In: *Natural Language & Linguistic Theory* 5 (3), pp. 403–439. ISSN: 1573-0859. DOI: 10.1007/BF00134555.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (2019). "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3645–3650. DOI: 10.18653/v1/P19-1355.

Strzyz, Michalina, David Vilares, and Carlos Gómez-Rodríguez (2019). "Viable Dependency Parsing as Sequence Labeling". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 717–723. DOI: 10.18653/v1/N19-1077.

Stucky, Susan U. (1987). "Configurational Variation in English". In: *Discontinuous Constituency*. Ed. by G.J. Huck and A.E. Ojeda. Vol. 20. Syntax and Semantics. Academic Press, pp. 377–404.

Suppes, Patrick (1972). "Probabilistic Grammars for Natural Languages". In: *Semantics of Natural Language*. Ed. by Donald Davidson and Gilbert Harman. Dordrecht: Springer Netherlands, pp. 741–762. ISBN: 978-94-010-2557-7. DOI: 10.1007/978-94-010-2557-7_25.

Szabolcsi, Anna (1989). "Bound Variables in Syntax (are there any?)" In: *Semantics and Contextual Expression*. Ed. by Renate Bartsch, J. F. A. K. van Benthem, and P. van Emde Boas. Dordrecht: Foris Publications, pp. 295–318.

Szántó, Zsolt and Richárd Farkas (2014). "Special Techniques for Constituent Parsing of Morphologically Rich Languages". In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 135–144. DOI: 10.3115/v1/E14-1015.

Tarski, Alfred (1955). "A lattice-theoretical fixpoint theorem and its applications". In: *Pacific journal of Mathematics* 5 (2), pp. 285–309.

Taskar, Ben, Dan Klein, Mike Collins, Daphne Koller, and Christopher D. Manning (2004). "Max-Margin Parsing". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, pp. 1–8. URL: https://www.aclweb.org/anthology/W04-3201.

Teichmann, Christoph, Alexander Koller, and Jonas Groschwitz (2017). "Coarse-To-Fine Parsing for Expressive Grammar Formalisms". In: *Proceedings of the 15th International Conference on Parsing Technologies (IWPT)*. Pisa, Italy, pp. 122–127. URL: https://www.aclweb.org/anthology/W17-6317.

Tesnière, Lucien (1959). *Éléments de syntaxe structurale*. Klincksieck. ISBN: 2-252-01861-5.

Thatcher, James W. (1967). "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory". In: *Journal of Computer and System Sciences* 1 (4), pp. 317–322. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(67)80022-9.

Thatcher, James W. and Jesse B. Wright (1968). "Generalized finite automata theory with an application to a decision problem of second-order logic". In: *Mathematical systems theory* 2 (1), pp. 57–81. ISSN: 1433-0490. DOI: 10.1007/BF01691346.

Turing, Alan M. (1950). "Computing Machinery and Intelligence". In: *Mind* LIX (236), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Versley, Yannick (2014a). "Experiments with Easy-first nonprojective constituent parsing". In: *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*. Dublin, Ireland: Dublin City University, pp. 39–53. URL: https://www.aclweb.org/anthology/W14-6104.

– (2014b). *Incorporating Semi-supervised Features into Discontinuous Easy-First Constituent Parsing*. arXiv: 1409.3813 [cs.CL].

– (2016). "Discontinuity (Re)^2-visited: A Minimalist Approach to Pseudoprojective Constituent Parsing". In: *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*. San Diego, California: Association for Computational Linguistics, pp. 58–69. DOI: 10.18653/v1/W16-0907.

Vieira, Tim and Jason Eisner (2017). "Learning to Prune: Exploring the Frontier of Fast and Accurate Parsing". In: *Transactions of the Association for Computational Linguistics* 5, pp. 263–278. ISSN: 2307-387X. URL: https://aclweb.org/anthology/Q17-1019.

Vijay-Shanker, Krishnamurti and David J. Weir (1994). "The Equivalence of Four Extensions of Context-free Grammars". In: *Math. Syst. Theory* 27 (6), pp. 511–546. ISSN: 0025-5661. DOI: 10.1007/BF01191624.

Vijay-Shanker, Krishnamurti, David J. Weir, and Aravind K. Joshi (1987). "Characterizing Structural Descriptions Produced by Various Grammatical Formalisms". In: *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*. Stanford, California, USA, pp. 104–111. DOI: 10.3115/981175.981190.

Vinyals, Oriol, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton (2015). "Grammar as a Foreign Language". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee,

M. Sugiyama, and R. Garnett. Curran Associates, Inc., pp. 2773–2781. URL: http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf.

Viterbi, Andrew J. (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* 13 (2), pp. 260–269. DOI: 10.1109/TIT.1967.1054010.

Wu, C. F. Jeff (1983). "On the Convergence Properties of the EM Algorithm". In: *The Annals of Statistics* 11 (1), pp. 95–103. ISSN: 00905364. URL: http://www.jstor.org/stable/2240463.

Wu, Pei-Chi (2004). "On Exponential-time Completeness of the Circularity Problem for Attribute Grammars". In: *ACM Trans. Program. Lang. Syst.* 26 (1), pp. 186–190. ISSN: 0164-0925. DOI: 10.1145/963778.963783.

Xia, Fei and Martha Palmer (2001). "Converting Dependency Structures to Phrase Structures". In: *Proceedings of the First International Conference on Human Language Technology Research*. URL: https://www.aclweb.org/anthology/H01-1014.

Yasunaga, Michihiro, Jungo Kasai, and Dragomir Radev (2018). "Robust Multilingual Part-of-Speech Tagging via Adversarial Training". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 976–986. DOI: 10.18653/v1/N18-1089.

Yeh, Alexander (2000). "More accurate tests for the statistical significance of result differences". In: *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics*. URL: https://www.aclweb.org/anthology/C00-2137.

Yi, Youngmin, Chao-Yue Lai, Slav Petrov, and Kurt Keutzer (2011). "Efficient Parallel CKY Parsing on GPUs". In: *Proceedings of the 12th International Conference on Parsing Technologies*. Dublin, Ireland: Association for Computational Linguistics, pp. 175–185. URL: https://www.aclweb.org/anthology/W11-2921.

Zeman, Daniel et al. (2019). *Universal Dependencies 2.5*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-3105.

Zhang, Hao and Ryan McDonald (2012). "Generalized Higher-Order Dependency Parsing with Cube Pruning". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 320–331. URL: https://www.aclweb.org/anthology/D12-1030.

Zhang, Xingxing, Jianpeng Cheng, and Mirella Lapata (2017). "Dependency Parsing as Head Selection". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia,

Spain: Association for Computational Linguistics, pp. 665–676. URL: https://www.aclweb.org/anthology/E17-1063.

Zhang, Yue and Stephen Clark (2011). "Syntactic Processing Using the Generalized Perceptron and Beam Search". In: *Computational Linguistics* 37 (1), pp. 105–151. DOI: 10.1162/coli_a_00037.

Zhu, Muhua, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu (2013). "Fast and Accurate Shift-Reduce Constituent Parsing". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 434–443. URL: https://www.aclweb.org/anthology/P13-1043.

A Appendix

A.1 Appendices to Chapter 3

The max-rule-product objective can be ill-defined. We give an example of a PRTG with chain rules where max-rule-product is an ill-defined parsing objective. Consider the PRTG $G^f = (N^f, \Sigma, S, R)$ with nonterminals $N^f = \{S, A_1, A_2, A_3\}$, terminals $\Sigma = \{\gamma, \alpha\}$, and rules with probabilities:

$$\begin{aligned} S &\to \gamma(A_1) && \#1.0 \\ A_1 &\to \gamma(A_2) && \#1.0 \\ A_2 &\to \gamma(A_3) && \#1.0 \\ A_3 &\to \alpha && \#1.0 \end{aligned}$$

Let $\mathcal{A}_S = (\{a\}^*, \cdot^{\mathcal{A}_S})$ be a string algebra where $\gamma^{\mathcal{A}_S}(x) = x$ for each $x \in \{a\}^*$ and $\alpha^{\mathcal{A}_S}() = a$. Consider the IRTG $\mathbb{G} = (G^f, \mathcal{A}_S, \mathrm{T}_\Sigma)$ and the chart G^f_a for the string a. Note that G^f_a is isomorphic to G^f because, for any $t \in L(G)$, we have $[\![t]\!]^{\mathcal{A}_S} = a$.

The inside and outside weight of each nonterminal of G^f_a is 1.0. If we merge A_1, A_2, and A_3 to A and project weights according to the max-rule-product principle, then we obtain the weighted RTG G^c_a with the following rules and weights:

$$\begin{aligned} S &\to \gamma(A) && \#1.0 \\ A &\to \gamma(A) && \#2.0 \\ A &\to \alpha && \#1.0 \end{aligned}$$

The weight of any tree in $L(G^c_a)$ is exponential in the number of occurrences of the chain rule $A \to \gamma(A)$ it contains. Consequently, there cannot be a best tree.

A.2 Appendices to Chapter 4

Proof of Theorem 4.3.7. By structural induction on t. Let $t = \sigma(t_1, \ldots, t_n)$ with

$$\mathrm{sort}_{\mathrm{align}}(\sigma) = ((k_1, s^1_1 \cdots s^1_{l_1}) \cdots (k_n, s^n_1 \cdots s^n_{l_n}), (k_0, (s^0_1 \cdots s^0_{l_0})))$$

and, for each $i \in [n]$, let

$$\begin{aligned}(u_1^i, \ldots, u_{k_i}^i) &= [\![t_i]\!]_{k_i}^{\mathcal{A}_1} \\ (\zeta_1^i, \ldots, \zeta_{l_i}^i) &= [\![t_i]\!]_{(s_1^i \cdots s_{l_i}^i)}^{\mathcal{A}_2} \\ (\lambda^i, \mu^i, \varrho^i, \alpha^i) &= [\![t_i]\!]_{(k_i, s_1^i \cdots s_{l_i}^i)}^{\mathcal{A}_3} \quad .\end{aligned}$$

Let $\sigma^{\mathcal{A}_1} = \langle w_1, \ldots, w_{k_0} \rangle$, $\sigma^{\mathcal{A}_2} = \langle \xi_1, \ldots, \xi_{l_0} \rangle$, and $\sigma^{\mathcal{A}_3} = \langle w_1, \ldots, w_{k_0}; \xi_1, \ldots, \xi_{l_0}; \alpha \rangle$, let $\varphi(x_r^q) = \zeta_r^q$ for each $q \in [n]$ and $r \in [l_q]$, and, for each $j \in [k_0]$, let $\mathrm{off}_{\mathrm{s}}^j$ be defined as in Definition 4.3.5.

(a) We show that:

$$\begin{aligned}\text{If} \quad & \forall i \in [n] \colon \forall j \in [k_j] \colon \lambda_j^i = |u_j^i| \ , \\ \text{then} \quad & \forall j \in [k_0] \colon \lambda_j^0 = |u_j^0| \ .\end{aligned} \tag{A.1}$$

Let $j \in [k_0]$. Note that $\forall m \in [|w_j|]_0$ it holds that

$$\begin{aligned}\mathrm{off}_{\mathrm{s}}^j(m) &= \sum_{m'=1}^{m} \begin{cases} 1 & \text{if } w_j[m'] \in \Gamma \\ \lambda_r^q & \text{if } w_j[m'] = x_r^q \end{cases} \\ &= \sum_{m'=1}^{m} \begin{cases} 1 & \text{if } w_j[m'] \in \Gamma \\ |u_r^q| & \text{if } w_j[m'] = x_r^q \end{cases} && \text{(by A.1)} \\ &= |(w_j[1..m])[x_r^q / u_r^q \mid q \in [n], r \in [k_q]]|\end{aligned}$$

Thus,

$$\begin{aligned}\lambda_j^0 &= \mathrm{off}_{\mathrm{s}}^j(|w_j|) = |(w_j[1..|w_j|])[x_r^q / u_r^q \mid q \in [n], r \in [k_q]]| \\ &= |w_j[x_r^q / u_r^q \mid q \in [n], r \in [k_q]]| = |u_j^0| \ .\end{aligned}$$ ■

(b) We show that:

$$\begin{aligned}&\text{If } \forall i \in [n] \colon \forall j \in [l_i] \colon \forall \xi_1', \ldots, \xi_{s_j^i}' \in \mathrm{U}_\Delta^*(Y) \colon \\ &\qquad \mu_j^i(\theta) = |\zeta_j^i[y_o / \xi_o \mid o \in [s_j^i]]| \quad \text{where } \theta(y_o) = \mathrm{len}(\xi_o') \text{ for each } o \in [s_j^i] \ , \\ &\text{then} \qquad \forall j \in [l_0] \colon \forall \xi_1', \ldots, \xi_{s_i^0}' \in \mathrm{U}_\Delta^*(Y) \colon \\ &\qquad \mu_j^0(\theta) = |\zeta_j^0[y_o / \xi_o' \mid o \in [s_j^0]]| \quad \text{where } \theta(y_o) = \mathrm{len}(\xi_o') \text{ for each } o \in [s_j^0] \ .\end{aligned} \tag{A.2}$$

This follows from

$$\begin{aligned}\mu_j^0(\theta) &= \sum_{j' \in \mathbb{N} \colon \to^{j'} \diamond \in \mathrm{pos}(\xi_j)} \mathrm{length}_{\theta,\mu}^{\xi_j}(\to^{j'} \diamond) \\ &= \mathrm{len}(\hat{\varphi}(\xi_j)[y_o / \xi_o' \mid o \in [s_j^0]]) && \text{(by Lemma A.2.1)} \\ &= \mathrm{len}(\zeta_j^0[y_o / \xi_o' \mid o \in [s_j^0]]) \ .\end{aligned}$$

Lemma A.2.1. $\forall p \in \mathrm{pos}(\xi_j)$: $\mathrm{length}^{\xi_j}_{\theta,\mu}(p) = \mathrm{len}(\hat{\varphi}(\xi_j|_p)[y_o/\xi'_o \mid o \in [s^0_j]])$. $\square$

The lemma is proved by induction on p (starting from leaf positions).

- Let $\xi_j|_p = y_o$. Then

$$\mathrm{len}(\hat{\varphi}(\xi_j|_p)[y_o/\xi'_o \mid o \in [s^0_j]]) = \mathrm{len}(\xi'_o) = \theta(y_o) = \mathrm{length}^{\xi_j}_{\theta,\mu}(p) \ .$$

- Let $\xi_j|_p = \delta(\bar{\xi})$. Then $\mathrm{len}(\hat{\varphi}(\xi_j|_p)[y_o/\xi'_o \mid o \in [s^0_j]]) = 1 = \mathrm{length}^{\xi_j}_{\theta,\mu}(p)$.
- Let $\xi_j|_p = x^q_r(\bar{\xi}_1, \ldots, \bar{\xi}_k)$ where $p = p'\diamond$. Then

$$\begin{aligned}
&\mathrm{len}(\hat{\varphi}(\xi_j|_p)[y_o/\xi'_o \mid o \in [s^0_j]]) \\
&= \mathrm{len}(\zeta^q_r[y_i/\hat{\varphi}(\bar{\xi}_i)[y_o/\xi'_o \mid o \in [s^0_j]] \mid i \in [s^q_r]]) \\
&= \mu^q_r(\hat{\theta}) \quad \text{where } \hat{\theta}(y_i) = \mathrm{len}(\hat{\varphi}(\bar{\xi}_i)[y_o/\xi'_o \mid o \in [s^0_j]) \qquad \text{(by A.2)} \\
&= \mu^q_r(\hat{\theta}) \quad \text{where } \hat{\theta}(y_i) = \sum_{j' \colon p'\, {}_i\!\searrow\!\to^{j'}\diamond \in \mathrm{pos}(\xi_j)} \mathrm{len}(\hat{\varphi}(\xi_j|_{p'\, {}_i\!\searrow\!\to^{j'}\diamond})[y_o/\xi'_o \mid o \in [s^0_j]) \\
&= \mu^q_r(\hat{\theta}) \quad \text{where } \hat{\theta}(y_i) = \sum_{j' \colon p'\, {}_i\!\searrow\!\to^{j'}\diamond \in \mathrm{pos}(\xi_j)} \mathrm{length}^{\xi_j}_{\theta,\mu}(p'\, {}_i\!\searrow \to^{j'} \diamond) \\
&\qquad\qquad \text{(by induction hypothesis of Lemma A.2.1)} \\
&= \mu^q_r(\psi^{\xi_j}_{\theta,\mu}(p)) = \mathrm{length}^{\xi_j}_{\theta,\mu}(p) \ . \qquad \blacksquare
\end{aligned}$$

(c) We show that:

If for each $i \in [n]$, $j \in [l_i]$, $\xi'_1, \ldots, \xi'_{s^i_j} \in \mathrm{U}^*_\Delta(Y)$, $\hat{o} \in [s^i_j]$:

$$\xi'_{\hat{o}} = \left(\zeta^i_j\left[y_o/\xi'_o \mid o \in [s^i_j]\right]\right)\big|^{\to^{\mathrm{len}(\xi'_{\hat{o}})}}_{\varrho^i_j(\theta)(\hat{o})} \tag{A.3}$$

where $\theta(y_o) = \mathrm{len}(\xi'_o)$ for each $o \in [s^i_j]$,
then, for each $j \in [l_0]$, $\xi'_1, \ldots, \xi'_{s^0_j} \in \mathrm{U}^*_\Delta(Y)$, and $\hat{o} \in [s^0_j]$:

$$\xi'_{\hat{o}} = (\zeta^0_j\left[y_o/\xi'_o \mid o \in [s^0_j]\right])|^{\to^{\mathrm{len}(\xi'_{\hat{o}})}}_{\varrho^0_j(\theta)(\hat{o})}$$

where $\theta(y_o) = \mathrm{len}(\xi'_o)$ for each $o \in [s^0_j]$.

Let $p \in \mathrm{pos}(\xi_j)$ be such that $\xi_j(p) = y_{\hat{o}}$. Then it holds that

$$\begin{aligned}
&(\zeta_j^0\left[y_o/\xi'_o \mid o \in [s_j^0]\right])|^{\to^{\mathrm{len}(\xi'_{\hat{o}})}}_{\varrho_j^0(\theta)(\hat{o})} \\
&= (\hat{\varphi}(\xi_j)\left[y_o/\xi'_o \mid o \in [s_j^0]\right])|^{\to^{\mathrm{len}(\xi'_{\hat{o}})}}_{\varrho_j^0(\theta)(\hat{o})} \\
&= \hat{\varphi}(\xi_j|_p)\left[y_o/\xi'_o \mid o \in [s_j^0]\right] && \text{(by Lemma A.2.2)} \\
&= \hat{\varphi}(y_{\hat{o}})\left[y_o/\xi'_o \mid o \in [s_j^0]\right] \\
&= y_{\hat{o}}\left[y_o/\xi'_o \mid o \in [s_j^0]\right] \\
&= \xi'_{\hat{o}}
\end{aligned}$$

Lemma A.2.2. For each $p \in \mathrm{pos}(\xi_j)$ it holds that

$$\hat{\varphi}(\xi_j|_p)\left[y_o/\xi'_o \mid o \in [s_j^0]\right] = \left(\hat{\varphi}(\xi_j)\left[y_o/\xi'_o \mid o \in [s_j^0]\right]\right)|^{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(p)}}_{\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p)} \quad .$$

□

Proof. By induction on p (starting from the root). For brevity we write $[y_o/\xi'_o]$ instead of $[y_o/\xi'_o \mid o \in [s_j^0]]$.

Induction base: If $p = \to^{i'} \diamond$, then

$$\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p) = \to^{(\mathrm{length}^{\xi_j}_{\theta,\mu}(<p))} = \to^{(\mathrm{length}^{\xi_j}_{\theta,\mu}(\to^0\diamond)+\ldots+\mathrm{length}^{\xi_j}_{\theta,\mu}(\to^{(i'-1)}\diamond))}$$

and, by Lemma A.2.1, we obtain that$\hat{\varphi}(\xi_j)[y_o/\xi'_o]$ equals

$$\underbrace{\underbrace{\hat{\varphi}(\xi_j|_{\to^0\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(\to^0\diamond)} \cdots \underbrace{\hat{\varphi}(\xi_j|_{\to^{i'-1}\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(\to^{(i'-1)}\diamond)}}_{\sum = \mathrm{length}^{\xi_j}_{\theta,\mu}(<p)} \underbrace{\hat{\varphi}(\xi_j|_{\to^{i'}\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(\to^{i'}\diamond)} \cdots \underbrace{\hat{\varphi}(\xi_j|_{\to^{\mathrm{len}(\xi_j)}\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(\to^{(\mathrm{len}(\xi_j)-1)}\diamond)}$$

and, thus,

$$\hat{\varphi}(\xi_j|_p)[y_o/\xi'_o] = \hat{\varphi}(\xi_j)[y_o/\xi'_o]|^{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(p)}}_{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(<p)}} \quad .$$

Induction step: If $p = p' \downarrow \to^{j'} \diamond$, then

$$\begin{aligned}
\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p) &= \mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p') \downarrow \to^{(\mathrm{length}^{\xi_j}_{\theta,\mu}(<p))} \\
&= \mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p') \downarrow \to^{(\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\downarrow\to^0\diamond)+\cdots+\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\downarrow\to^{(j'-1)}\diamond))} \quad .
\end{aligned}$$

Let $\xi_j|_{p'\diamond} = \delta(\zeta')$ for $\delta \in \Delta$, $\zeta' \in \mathrm{U}^*_\Delta(Y, X)$, and $z = \mathrm{len}(\zeta')$. Then

$$
\begin{aligned}
&\big(\hat{\varphi}(\xi_j)[y_o/\xi'_o]\big)\big|_{\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p'\diamond)} \\
&= \hat{\varphi}(\xi_j|_{p'\diamond})[y_o/\xi'_o] && \text{(induction hypothesis of Lemma A.2.2)} \\
&= \hat{\varphi}(\delta(\xi_j|_{p'\downarrow\to^0\diamond} \cdots \xi_j|_{p'\downarrow\to^{(z-1)}\diamond}))[y_o/\xi'_o] \\
&= \delta(\underbrace{\underbrace{\hat{\varphi}(\xi_j|_{p'\downarrow\to^0\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}((p'\downarrow\to^0\diamond))} \cdots \underbrace{\hat{\varphi}(\xi_j|_{p'\downarrow\to^{(j'-1)}\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\downarrow\to^{(j'-1)}\diamond)}}_{\sum = \mathrm{length}^{\xi_j}_{\theta,\mu}(<p)}
\end{aligned}
$$

$$
\underbrace{\hat{\varphi}(\xi_j|_{p'\downarrow\to^{j'}\diamond})[y_o/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\downarrow\to^{(j')}\diamond)} \cdots \underbrace{\hat{\varphi}(\xi_j|_{p'\downarrow\to^{(z-1)}\diamond})[y_o/\xi'_o])}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\downarrow\to^{(z-1)}\diamond)} \qquad \text{(Lemma A.2.1)}
$$

and, thus,

$$
\big(\hat{\varphi}(\xi_j)[y_o/\xi'_o]\big)\big|^{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(p)}}_{\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p')\downarrow\to^{(\mathrm{length}^{\xi_j}_{\theta,\mu}(<p))}} = \hat{\varphi}(\xi_j|_p)[y_o/\xi'_o] \ .
$$

Alternatively, let $p = p'_{i_1}\searrow\to^{i_2}\diamond$ where $\xi_j(p') = x^q_r$ and $i_1 \in [s^q_r]$. For clarity, we write m instead of s^q_r. There are $z_1, \ldots, z_m \in \mathbb{N}$ such that $i_2 \in [z_{i_1} - 1]_0$ and

$$
\xi_j|_{p'} = x^q_r(\xi_j|^{\to^{z_1}}_{p'_1\searrow}, \ldots, \xi_j|^{\to^{z_m}}_{p'_m\searrow}) \ .
$$

Moreover, by definition we have

$$
\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p) = \mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p'\diamond) \cdot \varrho^q_r(\psi^{\xi_j}_{\theta,\mu}(p'\diamond))(i_1) \cdot \to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(<p)} \ .
$$

It holds that

$$
\begin{aligned}
&\big(\hat{\varphi}(\xi_j)[y_o/\xi'_o]\big)\big|^{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\diamond)}}_{\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p'\diamond)} \\
&= \hat{\varphi}(\xi_j|_{p'\diamond})[y_o/\xi'_o] && \text{(induction hypothesis of Lemma A.2.2)} \\
&= \hat{\varphi}(x^q_r(\xi_j|^{\to^{z_1}}_{p'_1\searrow}, \ldots, \xi_j|^{\to^{z_m}}_{p'_m\searrow}))[y_o/\xi'_o] \\
&= (\zeta^q_r[y_i/\hat{\varphi}(\xi_j|^{\to^{z_i}}_{p'_i\searrow}) \mid i \in [m]])[y_o/\xi'_o] \\
&= \zeta^q_r[y_i/\hat{\varphi}(\xi_j|^{\to^{z_i}}_{p'_i\searrow})[y_o/\xi'_o] \mid i \in [m]] \ .
\end{aligned}
$$

Then, denoting $\mathrm{len}(\hat{\varphi}(\xi_j|^{\to^{z_{i_1}}}_{p'\,{}_{i_1}\searrow})[y_o/\xi'_o])$ by z', we have

$$\begin{aligned}
&\zeta^q_r[y_i/\hat{\varphi}(\xi_j|^{\to^{z_i}}_{p'\,{}_{i}\searrow})[y_o/\xi'_o] \mid i\in[m]]|^{\to^{z'}}_{\varrho^q_r(\psi^{\xi_j}_{\theta,\mu}(p'\diamond))(i_1)}\\
&= \hat{\varphi}(\xi_j|^{\to^{z_{i_1}}}_{p'\,{}_{i_1}\searrow})[y_o/\xi'_o] \qquad \text{(by A.3 because } z'=\psi^{\xi_j}_{\theta,\mu}(p'\diamond)(y_{i_1})\text{)}\\
&= \underbrace{\underbrace{\hat{\varphi}(\xi_j|_{p'\,{}_{i_1}\searrow\to^0\diamond})[y_0/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\,{}_{i_1}\searrow\to^0\diamond)}\cdots\underbrace{\hat{\varphi}(\xi_j|_{p'\,{}_{i_1}\searrow\to^{(i_2-1)}\diamond})[y_0/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\,{}_{i_1}\searrow\to^{(i_2-1)}\diamond)}}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(<p)}\\
&\qquad \underbrace{\hat{\varphi}(\xi_j|_{p'\,{}_{i_1}\searrow\to^{i_2}\diamond})[y_0/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\,{}_{i_1}\searrow\to^{i_2}\diamond)}\cdots\underbrace{\hat{\varphi}(\xi_j|_{p'\,{}_{i_1}\searrow\to^{(z_{i_1}-1)}\diamond})[y_0/\xi'_o]}_{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\,{}_{i_1}\searrow\to^{(z_{i_1}-1)}\diamond)} \qquad \text{(Lemma A.2.1)}
\end{aligned}$$

and thus

$$\begin{aligned}
&\big(\hat{\varphi}(\xi_j)[y_o/\xi'_o]\big)\big|^{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(p)}}_{\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p)}\\
&= \big(\hat{\varphi}(\xi_j)[y_o/\xi'_o]\big)\big|^{\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(p'\,{}_{i_1}\searrow\to^{i_2}\diamond)}}_{\mathbf{t}\mathrm{off}^{\xi_j}_{\theta,\mu,\varrho}(p'\diamond)\,\cdot\,\varrho^q_r(\psi^{\xi_j}_{\theta,\mu}(p'\diamond))(i_1)\,\cdot\,\to^{\mathrm{length}^{\xi_j}_{\theta,\mu}(<p)}}\\
&= \hat{\varphi}(\xi_j|_{p'\,{}_{i_1}\searrow\to^{i_2}\diamond})[y_0/\xi'_o]\ .
\end{aligned}$$

■

This concludes the proof of Theorem 4.3.7. ■

Proof of Theorem 4.3.8. By structural induction on t. Let $t=\sigma(t_1,\ldots,t_n)$, $\sigma^{\mathcal{A}_3}=\langle w_1,\ldots,w_{k_0};\xi_1,\ldots,\xi_{l_0};\alpha\rangle$ and, for each $i\in[n]$, let

- $(u^i_1,\ldots,u^i_{k_i})=[\![t_i]\!]^{\mathcal{A}_1}_{k_i}$,
- $(\zeta^i_1,\ldots,\zeta^i_{l_i})=[\![t_i]\!]^{\mathcal{A}_2}_{s^i_1\cdots s^i_{l_i}}$, and
- $(\lambda^i,\mu^i,\varrho^i,\alpha^i)=[\![t_i]\!]^{\mathcal{A}_3}_{(k_i,s^i_1\cdots,s^i_{l_i}}$.

We start the proof by an observation and two auxiliary lemmas. Then we make arguments about the domain and the codomain of α^0.

Observation A.2.3. Let $i\in[k_0]$ and $j,j'\in[|w_i|]_0$ with $j<j'$. Then $\mathrm{off}^i_{\mathrm{s}}(j)=\mathrm{off}^i_{\mathrm{s}}(j')$ implies $w_i[j']=x^q_r$ and $|u^q_r|=\varepsilon$ (or, equivalently, $\lambda^q_r=0$; see Theorem 4.3.7 (a)). □

Lemma A.2.4. Let $(i,j) \in \mathrm{dom}(\alpha)$, $\alpha(i,j) = (m,p)$, $\xi'_1, \ldots, \xi'_{s^0_m} \in \mathrm{U}^*_\Delta(Y)$, and $\theta\colon Y_{s^0_m} \to \mathbb{N}$ be such that $\theta(y_o) = \mathrm{len}(\xi'_o)$ for each $o \in [s^0_m]$. Then

$$\xi_m(p) = \zeta^0_m[y_o/\xi'_o \mid o \in [s^0_m]](\alpha^0_m(\theta)(i, \mathrm{off}^i_{\mathrm{s}}(j))) \ .$$

□

Proof. Let $\xi_m|_p = \delta(\xi')$ for $\delta \in \Delta$ and $\xi' \in \mathrm{U}^*_\Delta(Y, X)$. (Recall that $p \in \mathrm{pos}_\Delta(\xi_m)$.) Then $\mathrm{length}^{\xi_m}_{\theta,\mu}(p) = 1$ and

$$\begin{aligned}
&\zeta^0_m[y_o/\xi'_o \mid o \in [s^0_m]](\alpha^0_m(\theta)(i, \mathrm{off}^i_{\mathrm{s}}(j))) \\
&= \zeta^0_m[y_o/\xi'_o \mid o \in [s^0_m]](\mathbf{t}\mathrm{off}^{\xi_m}_{\theta,\mu,\varrho}(p)\ \diamond) \\
&= \hat{\varphi}(\xi^0_m)[y_o/\xi'_o \mid o \in [s^0_m]]|^{\rightarrow}_{\mathbf{t}\mathrm{off}^{\xi_m}_{\theta,\mu,\varrho}(p)}(\diamond) \\
&= \hat{\varphi}(\xi^0_m|_p)[y_o/\xi'_o \mid o \in [s^0_m]](\diamond) && \text{(Lemma A.2.2)} \\
&= \hat{\varphi}(\delta(\xi'))[y_o/\xi'_o \mid o \in [s^0_m]](\diamond) \\
&= (\delta(\hat{\varphi}(\xi')[y_o/\xi'_o \mid o \in [s^0_m]]))(\diamond) \\
&= \delta = \xi_m(p)
\end{aligned}$$

■

Lemma A.2.5. Let $q \in [n]$, $m \in [l_q]$, $\tau\colon Y_{s^q_m} \to \mathbb{N}$, $(i,j) \in \mathrm{dom}(\alpha^q_m(\tau))$, $i' \in [k_0]$, $j' \in [|w_{i'}|]$, $m' \in [l_0]$, and $p \in \mathrm{pos}(\xi_{m'})$ such that $x^q_i = w_{i'}[j']$ and $\xi_{m'}|_p = x^q_m(\xi'_1, \ldots, \xi'_{s^q_m})$. Let also $\xi''_1, \ldots, \xi''_{s^0_{m'}} \in \mathrm{U}^*_\Delta(Y)$ and $\theta\colon Y_{s^0_{m'}} \to \mathbb{N}$ be such that $\theta(y_o) = \mathrm{len}(\xi''_o)$ for each $o \in [s^0_{m'}]$. Then

$$\begin{aligned}
&\zeta^q_m[y_r/\psi(\xi'_r)[y_o/\xi''_o \mid o \in [s^0_{m'}]] \mid r \in [s^q_m]](\alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j)) \\
&= \zeta^0_{m'}[y_o/\xi''_o \mid o \in [s^0_{m'}]](\alpha^0_{m'}(\theta)(i', \mathrm{off}^{i'}_{\mathrm{s}}(j'-1)+j)) \ .
\end{aligned}$$

□

Proof.

$$\begin{aligned}
&\zeta^0_{m'}[y_o/\xi''_o \mid o \in [s^0_{m'}]](\alpha^0_{m'}(\theta)(i', \mathrm{off}^{i'}_{\mathrm{s}}(j'-1)+j)) \\
&= \zeta^0_{m'}[y_o/\xi''_o \mid o \in [s^0_{m'}]](\mathbf{t}\mathrm{off}^{\xi_{m'}}_{\theta,\mu,\varrho}(p) \cdot \alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j)) \\
&= \hat{\varphi}(\xi_{m'})[y_o/\xi''_o \mid o \in [s^0_{m'}]](\mathbf{t}\mathrm{off}^{\xi_{m'}}_{\theta,\mu,\varrho}(p) \cdot \alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j)) \\
&= (\hat{\varphi}(\xi_{m'})[y_o/\xi''_o \mid o \in [s^0_{m'}]]|^{\mathrm{length}^{\xi_{m'}}_{\theta,\mu}(p)}_{\mathbf{t}\mathrm{off}^{\xi_{m'}}_{\theta,\mu,\varrho}(p)})(\alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j)) \\
&= \left(\hat{\varphi}(\xi_{m'}|_p)[y_o/\xi''_o \mid o \in [s^0_{m'}]]\right)(\alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j)) && \text{(Lemma A.2.2)} \\
&= \left(\hat{\varphi}(x^q_m(\xi'_1, \ldots, \xi'_{s^q_m}))[y_o/\xi''_o \mid o \in [s^0_{m'}]]\right)(\alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j)) \\
&= \left(\zeta^q_m[y_r/\hat{\varphi}(\xi'_r)[y_o/\xi''_o \mid o \in [s^0_{m'}]] \mid r \in [s^q_m]]\right)(\alpha^q_m(\psi^{\xi_{m'}}_{\theta,\mu}(p))(i,j))
\end{aligned}$$

■

Domain of α^0: Let $i_1 \in [k_0]$ and $j_1 \in [|u^0_{i_1}|] = [\lambda^0_{i_1}]$.
Case 1: There is $j'_1 \in [|w_{i_1}|]$ such that $\mathrm{off}^{i_1}_{\mathrm{s}}(j'_1) = j_1$ and $w_{i_1}[j'_1] \in \Gamma$. By construction $(i_1, j_1) \in \mathrm{dom}(\alpha^0_m(\theta_m))$ where $m \in [l_0]$ is such that $\alpha(i_1, j'_1) = (m, p)$ for some $p \in \mathrm{pos}_\Delta(\xi_m)$. Note that if j'_1 exists, then j'_1 is unique because of Observation A.2.3.
Case 2: There is $j''_1 \in [|w_{i_1}|]$ such that $w_{i_1}[j''_1] = x^q_{i'_1}$ and $j_1 = \mathrm{off}^{i_1}_{\mathrm{s}}(j''_1 - 1) + j'_1$ where $q \in [n]$, $m \in [l_q]$, and $(i'_1, j'_1) \in \mathrm{dom}(\alpha^q_m(\tau))$ (for any $\tau \colon Y_{s^q_m} \to \mathbb{N}$). Then $(i_1, j_1) \in \mathrm{dom}(\alpha^0_{m'}(\theta_{m'}))$ for some $m' \in [l_0]$ by construction. Note that j''_1, q, m', i'_1, and j'_1 are unique if they exist because (b) and (c) hold for α^q by the induction hypothesis and because of Observation A.2.3.
Because of Observation A.2.3 either Case 1 or Case 2 applies. Consequently, (b) and (c) hold for α^0 and, for each $i \in [l_0]$, $\alpha^0_i(\theta)$ is a partial function.

Codomain of α^0: From what we showed about the domain of $\alpha^0_j(\theta_j)$, we know that each $(i', j') \in \mathrm{dom}(\alpha^0_j(\theta_j))$ was obtained such that either Lemma A.2.4 or Lemma A.2.5 applies. To see that α^0_j is injective consider the following argument: Suppose that there are $(i', j'), (i'', j'') \in \mathrm{dom}(\alpha^0_j(\theta))$ with $\alpha^0_j(\theta_j)(i', j') = \alpha^0_j(\theta_j)(i'', j'')$ and $i' \neq i''$ or $j' \neq j''$. Suppose that both pairs are obtained such that Lemma A.2.4 applies (with corresponding pairs (i'_o, j'_o) and (i''_o, j''_o) in $\mathrm{dom}(\alpha)$). Since α is injective, we have that $\alpha(i'_o, j'_o) \neq \alpha(i''_o, j''_o)$. Since the definition of $\alpha^0_j(\theta_j)$ depends only on the structure of ξ_j but not on the actual symbols from Δ at positions $\alpha(i'_o, j'_o)$ and $\alpha(i''_o, j''_o)$. Thus, by Lemma A.2.4 we have that $\xi_j(\alpha(i'_o, j'_o)) = \xi_j(\alpha(i''_o, j''_o))$, however, after changing one of the symbols, it would still hold that $\alpha^0_j(\theta_j)(i', j') = \alpha^0_j(\theta_j)(i'', j'')$, a contradiction.

Since, by induction the α^q_m are injective, one can argue similarly in the remaining cases using Lemma A.2.4 and Lemma A.2.5.

We argue that the codomain of $\alpha^0_j(\theta_j)$ is D: from Theorem 4.3.7 (c) we have that $\varrho^0_j(\theta_j)(\hat{o})$ is the position where the subhedge $\xi'_{\hat{o}}$ is located if substituted into ζ^0_j at $y_{\hat{o}}$. As Lemma A.2.4 and Lemma A.2.5 hold independently of the substituted tree, the domain of $\alpha^0_j(\theta_j)$ cannot contain a position of this subhedge. ■

A.3 Proof of Lemma 5.3.3

Proof of Lemma 5.3.3. By induction on ξ. Let $n \in \mathbb{N}$ and $\xi_1, \ldots, \xi_n \in \mathrm{U}_\Delta$ such that $\xi = \xi_1 \cdots \xi_n$. Then

$$\begin{aligned}&\left(\xi \ltimes \{(w_o, \to^{i_o}) \mapsto y_o \mid o \in [k]\}\right)[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]]\\ &= (u_0 v_1 u_1 \cdots u_{\kappa-1} v_\kappa u_\kappa)\,[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]]\end{aligned}$$

with κ, u_ℓ ($\ell \in [\kappa]_0$), and v_ℓ ($\ell \in [\kappa]$) as in Definition 5.3.2.

Let us first show that, for each $\ell \in [\kappa]$, we have

$$v_\ell[y_o/\xi|_{w_o}^{\to i_o} \mid o \in [k]] = \xi_{\hat{j}_\ell+1} \cdots \xi_{\hat{i}_\ell} \quad :$$

Case 1: $\exists o \in O_\ell$ such that $w'_o = \varepsilon$ and $i_o > 0$. Then for all $o' \in O_\ell \setminus \{o\}$, $w'_{o'} = \varepsilon$ and $i_{o'} = 0$ because the span position pairs are in parallel and $o' < o$ because of Definition 5.3.1 (c). Consequently, $\hat{i}_\ell = \hat{j}_\ell + i_o = j_o + i_o$. Thus $v_\ell = y_{o_1} \cdots y_{o_{\kappa'-1}} y_o$ with $O_\ell \setminus \{o\} = \{o_1, \ldots, o_{\kappa'-1}\}$, $o_x < o_{x+1}$ for each $x \in [\kappa' - 2]$, and

$$\begin{aligned}
&v_\ell[y_{o'}/\xi|_{w_{o'}}^{\to i_{o'}} \mid o' \in [k]] \\
&= \left(y_{o_1} \cdots y_{o_{\kappa'-1}} y_o\right)[y_{o'}/\xi|_{w_{o'}}^{\to i_{o'}} \mid o' \in [k]] \\
&= y_{o_1}[y_{o'}/\xi|_{w_{o'}}^{\to i_{o'}} \mid o' \in [k]] \ \cdots \ y_{o_{\kappa'-1}}[y_{o'}/\xi|_{w_{o'}}^{\to i_{o'}} \mid o' \in [k]] \ \ y_o[y_{o'}/\xi|_{w_{o'}}^{\to i_{o'}} \mid o' \in [k]] \\
&= \xi|_{w_{o_1}}^{\to 0} \cdots \xi|_{w_{o_{\kappa'-1}}}^{\to 0} \ \cdot \ \xi|_{w_o}^{\to i_o} \\
&= \varepsilon \ \cdots \ \varepsilon \ \cdot \ \xi|_{\to j_o}^{\to i_o} \\
&= \varepsilon \ \cdot \ \xi_{j_o+1} \cdots \xi_{j_o+i_o} \\
&= \varepsilon \ \cdot \ \xi_{\hat{j}_\ell+1} \cdots \xi_{\hat{i}_\ell} \ .
\end{aligned}$$

Case 2: $\nexists o \in O_\ell$ such that $w'_o = \varepsilon$ and $i_o > 0$. Let $\kappa'' \leq \kappa'$ be maximal such that there are $o_1, \ldots, o_{\kappa''} \in O_\ell$ where $w'_{o_x} = \varepsilon$ for each $x \in [k'']$ and $o_x < o_{x+1}$ for each $x \in [k''-1]$. Clearly, $i_{o_x} = 0$ for each $x \in [k'']$. Note that for all $o \notin O_\ell \setminus \{o_1, \ldots, o_{\kappa''}\}$ we have $w'_o \neq \varepsilon$ and thus $o > o_{\kappa''}$ because of Definition 5.3.1 (c). Thus, for each $x \in [\kappa'']$, we have $v_\ell^x = y_{o_x}$ and, for each $\kappa'' + 1 \leq x \leq \kappa'$, we have $v_\ell^x = \varepsilon$.

Case 2.1: If $\kappa' = \kappa''$, then $v_\ell = y_{o_1} \cdots y_{o_{\kappa''}}$ and $\hat{i}_\ell = 0 + \hat{j}_\ell$. Thus

$$\begin{aligned}
&v_\ell[y_{o'}/\xi|_{w_{o'}}^{\to i_{o'}} \mid o' \in [k]] \\
&= \xi|_{w_{o_1}}^{\to i_{o_1}} \cdots \xi|_{w_{o_{\kappa''}}}^{\to i_{o_1}} \\
&= \xi|_{w_{o_1}}^{\to 0} \cdots \xi|_{w_{o_{\kappa''}}}^{\to 0} \\
&= \varepsilon \\
&= \xi_{\hat{j}_\ell+1} \cdots \xi_{\hat{i}_\ell} \ .
\end{aligned}$$

Case 2.2: $\kappa'' < \kappa'$: Then there exists $o \in O_\ell$ with $w'_o \neq \varepsilon$. Hence,

$$\zeta = \sigma(\xi' \ltimes \{(w''_o, \to_{i_o}) \mapsto y_o \mid o \in O_\ell \colon w'_o =\downarrow w''_o\})$$

where $\xi_{\hat{j}_\ell+1} = \sigma(\xi')$. Thus $v_\ell = y_{o_1} \cdots y_{o_\kappa} \zeta$, $\hat{i}_\ell = 1 + \hat{j}_\ell$, and

$v_\ell[y_o/\xi|_{w_o}^{\to i_o} \mid o \in [k]]$

$$
\begin{aligned}
&= \xi|^{\to^{i_{o_1}}}_{w_{o_1}} \cdots \xi|^{\to^{i_{o_1}}}_{w_{o_\kappa}} \ \sigma(\xi' \ltimes \{(w''_o, \to_{i_o}) \mapsto y_o \mid o \in O_\ell \colon w'_o =\!\downarrow w''_o\} \\
&\qquad\qquad [y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]]) \\
&= \xi|^{\to^{i_{o_1}}}_{w_{o_1}} \cdots \xi|^{\to^{i_{o_1}}}_{w_{o_\kappa}} \ \sigma(\xi' \ltimes \{(w''_o, \to_{i_o}) \mapsto y_o \mid o \in O_\ell \colon w'_o =\!\downarrow w''_o\} \\
&\qquad\qquad [y_o/\xi'|^{\to^{i_o}}_{w''_o} \mid o \in O_\ell \colon w'_o =\!\downarrow w''_o]) \\
&= \xi|^{\to^{0}}_{w_{o_1}} \cdots \xi|^{\to^{0}}_{w_{o_\kappa}} \ \sigma(\xi') \qquad \text{(I.H.)} \\
&= \varepsilon \cdots \varepsilon \cdot \xi_{\hat{i}_j+1} \\
&= \xi_{\hat{j}_\ell+1} \cdots \xi_{\hat{i}_\ell} \ .
\end{aligned}
$$

It thus follows that

$$
\begin{aligned}
&(u_0 v_1 u_1 \cdots u_{\kappa-1} v_\kappa u_\kappa)\, [y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]] \\
&= u_0[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]] \qquad v_1[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]] \qquad u_1[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]] \\
&\quad \cdots\cdots \qquad\qquad v_\kappa[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]] \qquad u_\kappa[y_o/\xi|^{\to^{i_o}}_{w_o} \mid o \in [k]] \\
&= \xi_1 \cdots \xi_{\hat{j}_1} \qquad\qquad \xi_{\hat{j}_1+1} \cdots \xi_{\hat{i}_1} \qquad\qquad \xi_{\hat{i}_1+1} \cdots \xi_{\hat{j}_2} \\
&\quad \cdots\cdots \qquad\qquad \xi_{\hat{j}_\kappa+1} \cdots \xi_{\hat{i}_\kappa} \qquad\qquad \xi_{\hat{i}_\kappa+1} \cdots \xi_{\hat{j}_{\kappa+1}} \\
&= \xi_1 \cdots \xi_n = \xi
\end{aligned}
$$

■

Index

S-sorted set, 12
$\mathrm{SB}^{\ltimes}$, 95
hspan, 96
occ, 37
$\mathbf{t}$off, 77
n-best parsing, 61
p-value, 132
$\star_p$, 25
Π, 17
Σ-homomorphism, 14

algebra
  Σ-algebra, 14
  Σ-term algebra, 14
  product algebra, 16
alignment algebra, 78
alphabet, 9
  ranked, 16
approximate randomization, 132

below
  span position pair, 27
  stencil boundary, 96
bijective, 9
binarization, 113, 164
boundary, 97

chart, 21
child, 25
child labeling, 106
children, 25
coarse-to-fine parsing, 128
combinatory categorial grammar, 156
compatible, 16, 18
complexity
  hybrid grammar parsing, 122
  LCFRS parsing, 112
  sDCP parsing, 116
concatenation, 10
consistent
  weight assignment, 32
constituent tree, 28
context, 25
continuous, 29
convergent
  weight assignment, 32
crossing
  span position pairs, 26
  stencil boundary, 96

data-oriented parsing, 166
decomposable, 20
decomposition, 88
  hybrid grammar, 118
  LCFRS, 110
  sDCP, 114
dependency tree, 28
Dirichlet prior, 49
discontinuous, 29
domain, 14, 20

EM algorithm, 38, 44
encompass

span position pair, 26
stencil boundary, 96
expectation maximization algorithm, 38

F1-score, 124
family, 10
fanout
of a decomposition, 91
of a nonterminal, 68
of a set of integers, 91
of an LCFRS, 68
future length, 76

generalization, 57
grammar induction
corpus, 106
hybrid grammar, 103
LCFRS, 92
sDCP, 100
grammar morphism, 41

headed phrase structure grammar, 156
hedge
unranked, 22
hybrid grammar, 84
hybrid tree, 28

initial algebra semantics, 18
injective, 9
inside weight, 39
interpreted regular tree grammar, 19
probabilistic, 35
IRTG, *see* interpreted regular tree grammar

Kullback-Leibler divergence, 11

label
S-sorted tree, 13
unranked tree/hedge, 24
Lassy corpus, 133
latent annotation, 50, 167
LCFRS, *see* linear context-free rewriting systems
LCFRS algebra, 68
LCFRS/sDCP hybrid grammar, 84
length
unranked hedge, 23
lexicalized LCFRS, 104
lexicographic order, 10
likelihood, 12
linear context-free rewriting systems, 68, 164
linear functional grammar, 156
log-likelihood, 12

Markovization, 106
max-rule-product objective, 63, 205
merger, 54
Δ-merger, 55
mild context-sensitivity, 155
most probable parse, 35
most probable tree, 33

NeGra corpus, 132
non-crossing
stencil boundary, 96
non-overlapping
stencil boundary, 96
non-projective, 29
normalization, 37

operator tree, 13
oracle, 157
oracle reranker, 139
ordered parallel span position pairs, 94
outside weight, 39
overfitting, 57
overgeneration, 50
overlapping
span position pair, 26
stencil boundary, 96

parallel
span position pairs, 27

stencil boundary, 96
parent, 25
parsing problem, 20
part-of-speech tag, 130
partition, 10
phrase structure tree, 28
POS tag, 130
positions
S-sorted tree, 13
unranked tree/hedge, 24
precision, 124
probability assignment, 33
probability distribution, 11
conditional, 11
product of projections, 63
projective, 29
proper
weight assignment, 32
pseudo-projective, 158, 163

recall, 124
recursive partitioning, 88
directly extracted, 90
left-branching, 88
right-branching, 89
transformation, 93
regular tree grammar, 17
probabilistic, 32
regularly decomposable, 20
reranking objective, 62
robustness, 126
rule-projection algebra, 17

sampling, 61
sDCP, 72
sDCP algebra, 70
second-order substitution, 70
second-order variable, 22
significance, 132
simple definite clause program, 72
smoothing, 57
span
subhedge, 26
span position pair, 26
span positions
unranked tree/hedge, 26
split/merge algorithm, 50
split/merge cycle, 58
splitter, 52
2-splitter, 52
spurious ambiguity, 62
srk, 97
stencil boundary, 95
stencil operation, 94
strict labeling, 106
subhedge, 25, 26
subtree
unranked tree/hedge, 24
supertagging, 104
surjective, 9

tie breaking, 53
TiGer corpus, 132
transition system, 157
tree
S-sorted, 13
unranked, 22
tree adjoining grammar, 156
tree offset, 77
tree substitution grammar, 59, 166

unking, 129

variable, 11
Variational objective, 63
Viterbi objective, 60

weight assignment, 32